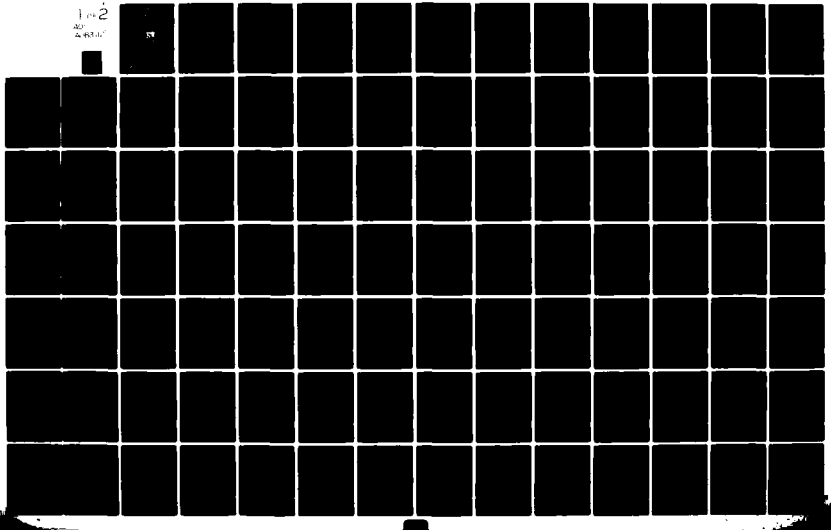
AD-A083 112 BOEING AEROSPACE CO SEATTLE WA BOEING MILITARY AIRPL--ETC F/G 9/2
COMPUTER PROGRAM DEVELOPMENT SPECIFICATION FOR IDAHST OPERATION--ETC(U)
NOV 76 F33615-76-C-1099
UNCLASSIFIED SPEC-SB-4041 AFAL-TR-76-208-ADD-1 NL



AFAL-TR-76-208, Addendum #1

SPECIFICATION
SB 4041

A055940

AD A047 163

LEVEL III

(1 inc)

(6) COMPUTER PROGRAM DEVELOPMENT SPECIFICATION
FOR IDAMST OPERATIONAL FLIGHT PROGRAMS
Addendum 1: EXECUTIVE SOFTWARE.

AD A083112

Prepared by

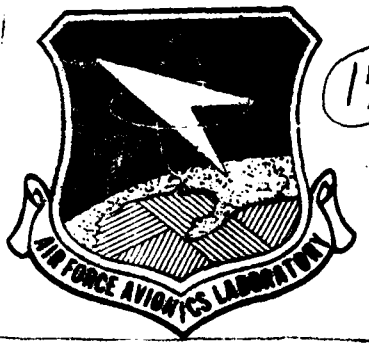
(12) 139

THE BOEING AEROSPACE COMPANY
BOEING MILITARY AIRPLANE DEVELOPMENT
SEATTLE, WASHINGTON

(11) NOV [redacted] 76

(18) AFAL (19) TR-76-208-ADD-1

(14) SPEC-SB-4041



(15) F33615-76-C-1099

PREPARED FOR

AIR FORCE AVIONICS LABORATORY
AIR FORCE SYSTEM COMMAND
UNITED STATES AIR FORCE
WRIGHT-PATTERSON AFB, OHIO 45433

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF TABLES

1.0 SCOPE

1.1 IDENTIFICATION

This specification establishes the requirements for performance and design of the Executive Software for the Integrated Digital Avionics for a Medium Short Takeoff and Landing Transport (IDAMST) system.

1.2 FUNCTIONAL SUMMARY

The IDAMST Executive provides the system software services which are utilized by the IDAMST Application Software. These services provide for the execution of real time applications, sharing of common data, interprocessor communication, and communication with and between Remote Terminals required to coordinate the operation of the core elements.

2.0 APPLICABLE DOCUMENTS

2.1 GOVERNMENT DOCUMENTS

2.1.1 Appendices to Contract F33615-76-C-1099.

a. Appendix A - "AMST Mission Profile and Scenario (Updated)".

b. Appendix C - "System Architecture".

c. Appendix E - "DAIS Mission Software, OFP Applications (SA-201-303)", 17 June 1976.

d. Appendix F - "DAIS Mission Software, Executive (SA-201-320)", 26 December 1975.

e. Appendix H - "Software Management Plan"

f. Appendix M - "TRW System Backup and Recovery Strategy (TRW 6404-5-6-06)", September 1975.

2.1.2 DAIS Documents (Reference)

a. ICD - Mission Operation Sequence: Pilot/Controls and Displays/Interface with Application Software (SA-803-200), 15 March 1976.

b. Mission Software/Controls and Displays Interface (SA-802-301), 12 March 1976.

c. DAIS System Control Procedures, (SA-100-101 Appendix A), 7 Nov. 1975.

2.1.3 IDAMST Documents (Program Generated)

a. Computer Program Development Specification, IDAMST OFP Applications (SB 4040-42), July 1976.

b. Computer Program Development Specification, IDAMST OFP Error Handling and Recovery (SB 4040-43), July 1976.

c. Computer Program Development Specifications, IDAMST Operational Test Program (SB 4040-44), July 1976.

2.1.4 IDAMST Documents (Reference)

The following documents because of release dates serve only as reference documentation for this specification; however, are considered prime to further definition of the IDAMST system design.

a. System Specification for IDAMST, Type A (S1-1010), June 1976.

b. Prime Item Development Specification, IDAMST Processor, Type B1 (S1 4030), June 1976.

c. System Segment Specification, IDAMST Control/Display Subsystem, Type A (Sl 5020), June 1976.

d. System Specification, IDAMST Information Transfer System, Type A (SS 3020), May 1976.

e. Prime Item Development Specification, IDAMST Remote Terminal, Type Bl (SS 3130), May 1976.

f. Prime Item Development Specification, IDAMST Bus Control Interface, Type Bl (SS 3230), May 1976.

2.2 NON-GOVERNMENT DOCUMENTS

a. Computer Sciences Corporation: Jovial J73/I Computer Programming Manual, October 1975.

b. Westinghouse Electrical Corporation: DAIS Processor Instruction Set, 1 November 1975.

c. DAIS Processor Prime Item Product Fabrication Specification (Preliminary) F33615-75-C-1154, December 1975.

d. DAIS Processor Engineering Data for Interface Control F33615-75-C-1154, March 1976.

e. C.S. Draper Laboratory: Interface Control Document PALEFAC, Pre-Processor/PALEFAC to Mission Software, January 1976.

3.0 REQUIREMENTS

The purpose of the IDAMST Executive is to isolate the physical aspects of the IDAMST federated system from the Application Software. The Executive allows the Application Software to reference time, Remote Terminals and information in other processors on a logical level. It masks the federated nature of the system, so that Application Software can be written as if it were to execute in a single, virtual machine. Finally, the IDAMST Executive controls the use of the Data Bus and provides mechanisms for error recovery.

The IDAMST Executive Software consists of two parts: a Local Executive and a Master Executive. Every processor in the IDAMST federated system contains a Local Executive; on the other hand, only one Master Executive is in operation at any given time. The Local Executive controls operations peculiar to a processor, including control of the Application Software within the processor and local participation in the I/O processes through the Data Bus. The Master Executive controls system-wide operations, including control of the Data Bus and system-wide error recovery.

3.1 PROGRAM DEFINITION

3.1.1 Hardware Interfaces

The IDAMST Executive interfaces with four elements of hardware: the IDAMST Multiplex System, the IDAMST Processor, the Mass Memory, and the Processor Control Panel.

3.1.1.1 The IDAMST Multiplex System

The IDAMST Multiplex System (commonly called the "Bus") is a series of hardware devices which permit communication between the various computers, displays and aircraft controls of the IDAMST system.

The Bus is dual redundant and consists of two twisted pairs. Messages may be sent across either bus; the second bus is used as a backup in case the system is unable to receive/transmit across the first.

The Bus rate is one megabit (i.e., it can transmit one million bits per second). The manner in which bits are encoded on the Bus is described in SA-301-200A (DAIS Digital, Command/Response, Time Division Multiplexing Data Bus; Section 2.1.b).

3.1.1.1.1 Bus Words

The basic parcel of information on the Bus is the 20 bit word. The first three bits are the SYNC bits. The SYNC bits serve two functions. SYNC bits inform the hardware to start reading a word. SYNC bits describe the format of the word which may be one of two types:

a. DATA Word
b. STATUS or COMMAND Word

The last bit is a parity bit. The rest of bits are information bits which depend upon the type (STATUS, COMMAND or DATA) of the word.

3.1.1.1.1.1 DATA Words

Data words consist of sixteen information bits and have two potential uses:

a. They may be data which will be written into a processor's memory.

b. They may follow a MODE COMMAND and their use will depend upon the MODE command. Only one or two data words will follow a MODE command.

3.1.1.1.1.2 STATUS Words

The STATUS word is transmitted in response to any command received by a terminal. Terminal in this case refers to any device capable of receiving and/or transmitting messages other than the Master Processor.

The format of the STATUS Word is:

Address	ME	Status	T/F

where:

a. Address is five bits containing the address of the terminal returning this status word.

b. Message Error is set to one if the last message sent was in error.

c. Status is nine bits which describe the internal status of the terminal. The meaning of these bits are terminal dependent.

d. T/F is set to one to indicate that the Master Processor should examine the Built In Test information available from this terminal.

3.1.1.1.1.3 COMMAND Words

A COMMAND word is sent to a terminal by the Master Processor whenever a terminal is to send a message, receive a message, perform a special function or inform the Master Processor of the terminal's status.

The format of the COMMAND word is:

Address	T/R	Sub Address	Word Count

where:

a. Address is five bits containing the address of the terminal to receive this message.

b. T/R is the transmit or receive bit. If this bit is set the terminal is to send a message. If this bit is not set, the terminal is to receive a message.

c. Sub Address is a five bit field used to identify the message to be sent or received. If this field has a value of zero, then this is a special type of command called a MODE command. A MODE command tells the terminal to perform some action other than send or receive a message.

d. Word count is a five bit field. For a command to send or receive data this will be the number of words transmitted. A word count of zero will be interpreted as a value of 32.

If the Command is a MODE COMMAND then this field contains the number for the MODE command. A MODE command of zero is always a request for Status. A MODE command may be followed by one or two data words which provide additional information to the terminal.

3.1.1.1.2 Data Bus Protocols

3.1.1.1.2.1 Master Transmission to a Terminal

When the Master Processor wishes to send a message to a Terminal:

a. The Master sends the terminal a Receive Command. The Receive Command contains the number of data words and the subaddress identifying the message being sent.

b. The Master sends the data words.

c. The Terminal transmits its status by sending a STATUS word.

3.1.1.1.2.2 Master Reception from a Terminal

When the Master Processor wishes to receive a message from a terminal:

a. The Master Processor sends a Transmit Command to the Terminal. The Transmit Command contains the number of data words and the subaddress identifying the message to be sent.

b. The Terminal sends a Status Command.

c. The Terminal sends the required number of data words.

3.1.1.1.2.3 Terminal Transmitting to a Terminal

When the Master Processor determines that a message is to be sent from a Terminal to a Terminal, the following actions are performed:

a. The Master Processor sends the Terminal which is to receive the message a Receive Command (as in 3.1.1.1.2.1).

b. The Master Processor sends the Terminal which is to transmit the message a Transmit Command (as in 3.1.1.1.2.2).

c. The Terminal which is to transmit a message sends its status (as in 3.1.1.1.2.2).

d. The Terminal which is to transmit a message sends the appropriate number of data words (as in 3.1.1.1.2.2).

e. The Terminal which is to receive reads the data words (as in 3.1.1.1.2.1).

f. The Terminal which is to receive sends its Status (as in 3.1.1.1.2.1).

g. Only the Master Processor receives the status words and checks the correct functioning of the terminals.

3.1.1.1.2.4 Terminal Desiring to Transmit or Receive

If a terminal wishes to transmit or receive a message, it sets the appropriate bit in its Status Word.

The next time the Status Word is transmitted, the Master Processor will note that a message is to be sent. When the Master Processor is ready, it will read either the Activity Register or the Status Word of the Terminal to determine which message the terminal wishes to send, and to whom the message will be sent.

3.1.1.1.2.5 Time-Outs

When either Master Processor or a Terminal expect to receive data, there will be a wait of 75 microseconds. If within that time the Bus Interface has not seen data words on the Bus, then the Processor or Terminal Bus Interface times-out and assumes that no data is going to be transmitted. At this time a Terminal will respond with its STATUS word signalling a Message Error.

3.1.1.1.3 The Bus Control Interface Unit

3.1.1.1.3.1 Definition

The Bus Control Interface Unit (BCIU) shall provide the interface control and data transfer function required to connect a Processor with two multiplexed data buses. The BCIU shall be directed to operate in a mode by its interfacing processor. The following are the modes in which the BCIU shall be capable of operating.

a. Remote Mode, providing transfer of data in both directions between the Processor and either of the two Buses, providing status replies on the appropriate bus in response to commands, and special internal operations and interrupts to the associated processor upon receipt of certain special commands on the data buses.

b. Master Mode, providing control of the data bus based upon instructions fetched from the memory of the Processor through the Direct Memory Access (DMA) Channel by the BCIU.

This Master Control mode shall result in:

1. The BCIU issuing Bus Commands to other devices on the Data Buses.

2. Participating in data transfers on the buses (when the instruction dictates it).

3. Checking status responses from devices on the data buses.

4. Checking formats of the data bus operation.

5. Reporting of error conditions to the processor.

At any time, there shall only be one BCIU in Master Mode.

3.1.1.1.3.2 BCIU Registers

The Registers of the BCIU control its mode of operation, provide information for the Master Processor and provide information to its local processor.

BCIU registers are accessed through the PIO instruction (to be defined in the IDAMST Processor Instruction Manual) by an address as given in Table 3.1.1.1.3.2-1. The meaning and format of each register is discussed in the Bus Control Interface Unit B-1 Specification SA 301300B.

PIO ADDRESS	REGISTER
0	PROCESSOR CONTROL REGISTER (PCR)
1	INTERNAL STATUS REGISTER (ISR)
2	BASE ADDRESS REGISTER (BAR)
3	INSTRUCTION ADDRESS REGISTER (IAR)
4	BUILT-IN-TEST REGISTER (BITR)
5	MODE DATA REGISTER (MDR)
6	LAST COMMAND REGISTER (LCR)
7	STATUS CODE REGISTER (SCR)
8	MASTER FUNCTION REGISTER (MFR)
9	POINTER REGISTER (PR)
10	DATA ADDRESS REGISTER (DAR)
11	WORD COUNT REGISTER (WCR)
12	XMIT STATUS WORD REGISTER (XSWR)
13	RECV STATUS WORD REGISTER (RSWR)
14	INSTRUCTION WORD REGISTER 1 (IWR1)
15	INSTRUCTION WORD REGISTER 2 (IWR2)

Table 3.1.1.1.3.2-1 BCIU REGISTERS (PIO ACCESSIBLE)

3.1.1.1.3.2.1 Processor Control Register (PCR)

This register (format shown below) shall contain indicators which generally control the BCIU actions and in certain instances reflect a particular BCIU state.

At the time power is applied (including during a transient recovery), the BCIU will clear the PCR (and also the ISR), perform a self test, and present a power up interrupt (level 1) to the processor.

The format of the PCR is:

1	2	3	4	5	6	7	11	12	13	14	15	16
MASTER	GO	FAIL	SPARE	SRX	STBYP	BCIU ADDRESS		SPARE	READY	SPARE	BUSY CONT.	RUN

a. Master- This bit shall be set to logic 1 by the Processor, in conjunction with the GO bit, to direct the BCIU to operate in Master Mode. Conversely, the bit shall be set to logic 0 to indicate Remote Mode.

b. Go - This bit shall be set to logic 1 by the Processor to indicate the BCIU is to enter an operational mode. The bit shall be set to logic 0 by the Processor to indicate an operational mode is to be terminated. The bit shall be set to logic 0 by the BCIU (Master Mode only) after an operational mode is terminated by either a HALT instruction or an unrecoverable direct memory access error condition.

c. Fail - This bit shall be set to logic 1 if the BCIU detects an error in self-test during the power-on initialization. The READY shall also be set to logic 1. In Master Mode, the BCIU shall set the FAIL bit (level 6 Interrupt) to indicate a Failure.

d. Spare - This bit shall be a logic 0 and available for future use.

e. System Reset Acknowledge - This bit shall be set to logic 1 by the Processor to indicate acknowledgement of the power-on-reset interrupt.

f. Self Test by-Pass - This bit shall be set to a logic 1 by the processor to indicate that the BCIU is not to perform self-test.

g. BCIU Address - These 5 bits shall be set by the Processor, upon initially directing the BCIU to enter an operational mode, to indicate the address of the BCIU.

h. Spare - This bit shall be a logic 0 and available for future use.

i. Ready - This bit shall be set to logic 1 by the BCIU after completing its power-on initialization.

j. Spare - This bit shall be a logic 0 and available for future use.

k. Busy/Cont - In Remote Mode, the bit is set to logic 1 by the BCIU after any interrupt is presented to the Processor in order to indicate BCIU Busy State entered. The bit is set to 1 by the Remote Processor to indicate BCIU is to enter Processor initiated Busy State. It is set to logic 0 by BCIU after having been directed to Exit Busy State by the Remote Processor or via the Busy Override Mode Command from the Master Processor.

In Master Mode, the bit is set to logic 1 by the Master Processor to indicate to the Master BCIU that an interrupt has been processed and the BCIU is to continue normal operation. The BCIU shall set this bit to logic 0 prior to entering the Master Mode Pseudo Wait State.

l. Run - This bit shall be set to logic 1 after Processor directs BCIU to enter an operational mode or upon exiting a Busy state. The bit shall be set to logic 0 by the BCIU after an operational mode has been terminated.

3.1.1.1.3.2.2 Internal Status Register (ISR) - This register shall contain indicators which are set only by the BCIU. These indicators shall generally reflect any condition detected by the BCIU during the last bus or internal operation which warrants an interrupt to the associated Processor. The register shall be cleared by BCIU prior to processing a new instruction/command.

The format of the ISR is:

	1															16
	H	PCI	IVI/SI	MDP	AXR	AM	MF	XSEX	RSEX	XSE	RSE	NDR	ICD	DPE	IVD	DMA
Interrupt Level	Level 1			Level 2			Level 3					Level 4				Level 5

The meaning of each bit is:

Bit	Symbol	Meaning
1	H	Halt
2	PCI	Program Controlled Interrupt
3	IVI/SI	Invalid Instruction (Master Mode) System Interrupt
4	MDP	Mode Data Present
5	AXR	Async Msg Int 0=RECV 1=XMIT
6	AM	Async Msg Int
7	MF	Master Function (Remote only)
8	XSEX	XMIT STATUS Exception
9	RSEX	Recv Status Exception
10	XSE	XMIT Status Error
11	RSE	Recv Status Error
12	NDR	No Data Received
13	ICD	Incomplete Data
14	DPE	Data Parity Error
15	IVD	Invalid Data
16	DMA	Direct Memory Access (Error)

The above bits will only be set for given interrupts.

The meaning of each bit is:

a. HALT (H) - This bit shall be set to logic 1, in Master Mode only, to indicate that the BCIU has processed a HALT instruction. The operational mode (Master) shall be terminated.

b. Program Controlled Interrupt (PCI) - This bit shall be set to logic 1, in Master Mode only, after completion of 2 word instruction operation in which PCI was requested (PCI=1).

c. Invalid Instruction (IVI) - In Master Mode only, this bit shall be set to logic 1 if the Device Address within the Receive field of the 2-word instruction is equal to the Device Address within the Transmit field.

d. System Interrupt (SI) - In Remote Mode only, this bit shall be set to logic 1 upon recieving the System Interrupt Mode Command.

e. Mode Data Present (MDP) - This bit shall be set to logic 1, in Master Mode only, after successfully receiving the Data Word associated with Mode Operations (Interrupt results from mode operations 3, 10, 11 and 13).

f. Asynchronous Message Xmit/Recv (AXR) - In Master or Remote Modes, this bit shall be set in conjunction with Bit 6 (AM) to indicate whether the BCIU was the Receiver (AXR=0) or the Transmitter (AXR=1) of an asynchronous message (Sub-Address=31).

g. Asynchronous Message (AM)- In Master or Remote Modes, this bit shall be set to logic 1 after successful completion of an asynchronous bus message operation (Sub-Address=31).

h. Master Function (MF) - This bit shall be set to logic 1, in Remote Mode only, after receiving the Master Function Mode Command (usually called the Minor Cycle Event).

i. Transmit Status Exception (XSEX) - This bit shall be set to logic 1, in Master Mode only, after receiving an excepted, valid status word associated with a Remote device in Transmit Mode in which the Message Error, Terminal Failure, or Status Code is non-zero. The status word shall be placed intact within the XMIT Status Word Register.

j. Receive Status Exception (RSEX) - This bit shall be set to logic 1, in Master Mode only, after receiving an expected, valid status word associated with a Remote device in Receive Mode in which the Message Error, Terminal Failure, or Status Code is non-zero. The status word shall be placed intact within the Received Status Word Register.

k. Transmit Status Error (XSE) - This bit shall be set to logic 1, in Master Mode only, if an expected status word associated with a Remote Device in Receive mode, is not received, is received invalidly, is received validly with bad parity, or is received validly with good parity with a Device Address that does not match the Receive Device Address within the 2-word instruction.

l. Receive Status Error (RSE)- This bit shall be set to logic 1, in Master Mode only, if an expected status word associated with a Remote Device in Receive mode, is not received, is received invalidly, is received validly with bad parity, or is received validly with good parity with a Device Address that does not match the Receive Device Address within the 2-word instruction.

m. No Data Receive (NDR) - This bit shall be set to logic 1, in Master Mode only, after commanding a Remote device to transmit one or more data words and the first such data word has not arrived within 60 microseconds after status word reception.

n. Incomplete Data (ICD) - This bit shall be set to logic 1, in Master Mode only, after receiving at least one expected data word and with further data words expected, the next data word is not received within 60 microseconds after reception of the last data word.

o. Invalid Data (IVD) - This bit shall be set to logic 1, in Master Mode only, after an expected data word was received with Parity Error indicated. Data word reception continues.

p. Data Parity Error (DPE) - This bit shall be set to logic 1, in Master Mode only, after an expected data word was received with Parity Error indicated. Data word reception continues.

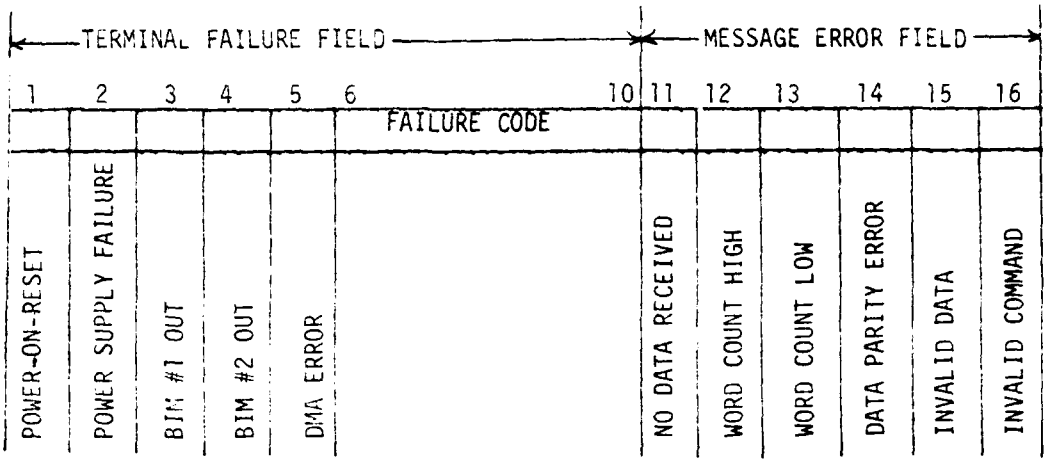
q. Direct Memory Access Error (DMA)- This bit shall be set to logic 1, in Master or Remote Mode, after an unrecoverable DMA Error is detected while attempting to fetch an instruction word, a pointer word, or a data word from main memory or while attempting to store a tag word or a data word into main memory.

3.1.1.1.3.2.3 Base Address Register (BAR)- This register shall be set only by a Processor for the associated BCIU (Master/Remote) and shall contain the most significant 10 bits of a pointer word address within main memory for a given data transfer operation. The addressed pointer word shall contain the true data block address.

3.1.1.1.3.2.4 Instruction Address Register (IAR)- This register shall be set only by a Processor whose associated BCIU is to operate in Master Mode. The register shall contain the main memory address of the initial 2-word instruction is executed, the BCIU shall modify the register in order to reflect the address of the next instruction to be executed. The register shall be unused in Remote Mode.

3.1.1.1.3.2.5 Last Command Register (LCR) - This register shall be used only in support of the Transmit Last Command Mode Command. In Remote mode, the BCIU shall place commands which are received validly and directed to the particular BCIU into this register. Exceptions shall be Transmit Status Word, Transmit Bit Word, and the Transmit Last Command itself.

3.1.1.1.3.2.6 Built-In Test Word Register (BITR) - This register shall be used to either maintain the Built-In Test Word (Remote Mode), or to temporarily hold Terminal Failure or bus monitoring of own transmission information (Master Mode). The format of a BCIU BIT word is shown in the figure below and described in the following paragraphs.



a. Power-On-Reset - This bit shall be set to logic 1 if the BCIU performs Power-On Initialization.

b. Power Supply Failure - This bit shall not be implemented for the BCIU (Set to Logic 0).

c. BIM 1 Out - This bit shall be set to logic 1 by the Remote Mode BCIU after powering down Bus Interface Module (BIM) 1 as a result of receiving a Remove Power BIM 1 Mode Command. The BIT shall indicate that power has been removed from BIM 1.

d. BIM 2 Out - This bit shall be set to logic 1 by the Remote ode BCIU after powering down BIM 2 as a result of receiving a Remove Power BIM 2 Mode Command. The bit shall indicate that power has been removed from BIM 2.

e. DMA Error - This bit shall be set to logic 1 by the Remote Mode BCIU after an unrecoverable direct memory access error is detected while fetching data words from or storing data words (excluding tag words) into main memory.

f. Failure Code Errors - The Failure Code shall be set for the following Remote BCIU errors: BIM Failure, BCM (Bus Control Module) ROM Parity error, BCM Data Flow error. The BCIU in master mode shall indicate the BIM and BCM Data Flow Codes.

g. No Data Received - This bit shall be set to logic 1 by the Remote Mode BCIU after having been directed to receive one or more data words and the first such data word has not arrived within 75 microseconds after command word reception.

h. Word Count Low - This bit shall be set to logic 1 by the Remote Mode BCIU after having been directed to receive two or more data words, at least one such data word has arrived, but the next expected data word does not arrive within 60 microseconds of last data word receptio

i. Word Count High - This bit shall be set to a logic 1 by the Remote Mode BCIU after detecting another Data Word after the word count is zero.

j. Data Parity Error - This bit shall be set to logic 1 by the Remote BCIU after an expected data word was received with Parity Error indicated. Data word reception continues.

k. Invalid Data - This bit shall be set to logic 1 by the Remote Mode BCIU after an expected data word was received with RECV WORD INVALID indicated. Data word reception continues.

l. Invalid Command - This bit shall be set to logic 1 by the Remote BCIU after receiving a mode command in which the mode code designates an invalid operation for the BCIU.

3.1.1.1.3.2.7 Status Code Register (SCR) - This register shall be used in Remote Mode only and shall be set and reset by the Remote Mode Processor. The actual status code shall be the nine (9) least significant bits of the register and shall be merged into any status word prior to status word bus transmittal by the Remote BCIU.

3.1.1.1.3.2.8 Master Function Register (MFR) - This register shall be used only in support of the Master Function Mode Command. In Master Mode and in accordance with Master Function processing, the contents of the register shall be transmitted to the Remote device as a data word immediately following the command word. It shall be the Master Processor's responsibility to set the register. In Remote Mode, the Remote Mode BCIU shall place the received data word, in response to the Master Function mode command, into the Master Function Register. It shall be the Remote Processor's responsibility to then interpret the contents of the register.

3.1.1.1.3.2.9 Instruction Word Register 1 (IWR1) - This register shall be used in Master Mode only to hold the first half of the current 32-bit instruction.

3.1.1.1.2.3.10 Instruction Word Register 2 (IWR2) - This register shall be used in Master Mode only to hold the second half of the current 32-bit instruction.

3.1.1.1.3.2.11 Xmit Status Word Register (XSWR) - This register shall be used in Master Mode only to hold any status word received from a Remote Device in Transmit Mode, in which the Message Error, Terminal Failure, or Status Code fields were non-zero.

3.1.1.1.3.2.12 Received Status Word Register (RSWR) - This register shall be used in Master Mode only to hold any status word recieved from a Remote device in Receive Mode, in which the Message Error, Terminal Failure, or Status Code fields were non-zero.

3.1.1.1.3.2.13 Mode Data Register (MDR) - In Master Mode, and only in accordance with performing a certain class of mode commands, the contents of this register shall be transmitted to the Remote device as a data word immediately following the command word. The Master Processor shall be responsible for setting the register.

In Remote Mode, the MDR shall be undefined for the Mode Operations defined.

3.1.1.1.3.2.14 Pointer Register (PR) - This register shall be set by a BCIU operating in either Master or Remote mode and shall contain the initial data area address for a given data bus operation involving main memory data transfers. The register shall be used in Tag Word Operations.

3.1.1.1.3.2.15 Data Address Register (DAR) - This register shall be set by a BCIU operating in either Master or Remote mode and shall be used to indicate the main memory address of the next data word to be fetched/stored in support of a given bus operation. The register shall be derived from the Pointer Register and in all cases (Receive or Transmit) that value shall be initially incremented by 1 to get over the Tag Word. This value then becomes the address to fetch/store the first data word. As each word is fetched/stored, the BCIU shall increment the register value by 1 to affect sequential data word fetch/stores.

3.1.1.1.3.2.16 Word Count Register (WCR) - This register shall be derived from the Bus Command and set by the BCIU in either Master or Remote Mode. In Bus Operations involving data word transfers, it shall indicate the remaining number of data words to be transferred. The register shall be decremented by 1, by the BCIU, as each data word transfer is performed.

3.1.1.1.3.2.17 The BCIU Command List - The Master Processor is the processor attached to the BCIU which is operating in Master Mode. The Master Processor is the only processor which is capable of initiating message transmission. No other processor, BCIU or RT sends messages until it has been told to do so by the Master BCIU and Master Processor.

Each BCIU command consists of two words in the following format:

Bits 1 2	3 4	5	6	7 11	12 16
OP CODE	RETRY	SPARE	I	RECEIVE DEVICE ADDRESS	RECEIVE SUBADDRESS / MODE
WORD COUNT/MODE CODE			B	TRANSMIT DEVICE ADDRESS	TRANSMIT SUBADDRESS / MODE

Each of the fields in the two word instruction have the following uses:

a. OP CODE - These two bits determine the function of the command.

00 = Halt the BCIU. This is always the last command in a list and denotes that no other command is to be performed. When the BCIU executes this instruction the Halt bit is set in the Internal Status Register and a BCIU level 1 interrupt will be generated.

01 = No Operation. This OP code has two uses. The first is to cancel commands when the Master Processor no longer wishes the Master BCIU to perform.

The second is to create a processor interrupt before the next BCIU instruction is generated. A typical use of the latter case is sending Mode Commands. The Mode Data Register must be set before the command is sent. Thus, the interrupt causes a BCIU pause which permits the Master Processor to set the MDR and then set the Continue Bit in the PCR to resume BCIU processing.

For this OP code only the interrupt field is examined. All other options are ignored.

11 = Bus Operation. For this operation the rest of the fields are interpreted as reception or transmission across the Bus.

b. RETRY - If the transmission attempted by this instruction was not successfully completed, and this field is not zero, then the transmission will be retried up to three times.

c. SPARE - This bit is not used.

d. I - If this bit is set, successful completion of this instruction will cause an interrupt. The PCI bit in the ISR will be set. The interrupt will be of level 1. The discussion accompanying the No Operation Code explains the use of this bit, although the bit may be used in any of the four instructions.

e. RECEIVE DEVICE ADDRESS - This field contains the address of the terminal to receive the message. This field is only used for BCIU instruction OP code "Bus Operation". If the Receive Device Address is not the address of the Master BCIU (as contained in the BCIU address field of the PCR), then a Receive Command will be formed by concatenating the Receive Device Address Field, a bit denoting Receive, the Receive Subaddress/Mode field, and the Word Count/Mode Code field. This receive command will then be transmitted across the Bus.

If the Receive Device Address field is the address of this BCIU and the Receive Subaddress/Mode field is not zero (i.e., this is not a Mode Command), then the Receive Subaddress field will be used to load the Data Address Register (see Section 3.1.1.1.3.2.15) which will then determine where the received message will be stored.

f. RECEIVE SUBADDRESS/MODE - This field describes the message to be received. The use of this field is described in the Receive Device Address field. If this address were zero it would indicate that this is a Mode Command.

g. WORD COUNT/MODE CODE - For mode commands this field contains the number of the command. For Receive/Transmit commands this field contains the number of data words to be transmitted. Mode commands are discussed in 2.1.1.1.3.2.18.

h. B - This field indicates which Bus will be used for data transmission. If this bit is zero, Bus number one will be used. If this bit is one, Bus number two will be used.

i. TRANSMIT DEVICE ADDRESS - This field is similar to the Receive Device Address except that it is the address of the terminal which will send the message.

If the address is not that of the Master BCIU, then Transmit Command will be formed by concentrating the Transmit Device Address field, the Transmit bit, the Transmit Subaddress/Mode field and the Word Count/Mode Code field.

If the Transmit Device Address field is the address of this terminal then the Data Address Register will be formed (see Section 3.1.1.1.3.2.15) and the data will be written into the Bus from that address.

For Mode Commands the Transmit Device Address field is the address of this terminal then the Data Address Register will be formed (see Section 3.1.1.1.3.2.15) and the data will be written into the Bus from that address.

For Mode Commands the Transmit Device Address field is the address of the terminal to receive the Mode Command and the Receive Device Address field is the address of the Master BCIU.

It is an error for the Receive Device Address field and the Transmit Device Address field to be the same device. This error will cause an interrupt of level 1 and the IVI bit will be set in the Internal Status Register.

j. TRANSMIT SUBADDRESS/MODE - The use of this field has been discussed in the description of the Transmit Device Address field.

For mode commands, both the Transmit Subaddress and Receive Subaddress will be zero.

3.1.1.1.3.2.18 Mode Commands - The available Mode Commands are described in the table below. Mode Commands beyond these are to be used for system expansion and are presently undefined. For these undefined Mode Commands, the only response of the terminal is to transmit its status word. The software interface is given in Figure 3.1.1.1.3.2.18-1 for both the BCIU and remote terminals.

The BCIU determines whether a given Mode Command also requires data transmitted to or received from a terminal. In the case that data should be transmitted the Mode Data register will be written. In the case that it should be read the data received will be stored in the Mode Data register.

MODE CODE NUMBER	BCIU (REMOTE MODE)	BCIU (MASTER MODE)
0	Invalid	Invalid
1	Transmit Status Word	Transmit Status Word
2	Invalid	Reset Status Code Field
3	Transmit BIT Word	Transmit BIT Word
4	Remove Power Bus Interface 1	Remove Power Bus Interface 1
5	Remove Power Bus Interface 2	Remove Power Bus Interface 2
6	Shutdown Override Bus Interface 1	Shutdown Override Bus Interface 1
7	Shutdown Override Bus Interface 2	Shutdown Override Bus Interface 2
8	Invalid	Initiate Terminal Self Test
9	Initialize Terminal	Initialize Terminal
10	Transmit Last Command	Transmit Last Command
11	Invalid	Interrogate Activity Register
12	Invalid	Reset Serial Input Channel
13	Invalid	Interrogate Module Error Register
14	Invalid	Initiate Serial Channel I/O
15	Invalid	Word Masking
16	Invalid	Bit Masking
17	No-op	No-op
18	Master Function Interrupt	Master Function Interrupt
19	Invalid	Valid (Status)*
20	Busy Override	Busy Override
21	System Interrupt	System Interrupt
22	Invalid	Valid (Status)*
23	Invalid	Valid (Status)*
24	Invalid	Valid (Status)*
25	Invalid	Valid (Status)*
26	Invalid	Valid (Status)*
27	Invalid	Valid (Status)*
28	Invalid	Valid (Status)*
29	Invalid	Valid (Status)*
30	Invalid	Valid (Status)*
31	Invalid	Valid (Status)*

* Valid (Status) - BICU in Master Mode shall expect only a Status Word Response for all undefined Mode Operations.

Table 3.1.1.1.3.2.18-1 BCIU Mode Operations

Numbers 22 through 31 are Undefined commands.

Details of the uses of each of these mode codes are described fully in the BCIU Specification.

Mode Codes 0, 1, 2, 3, 8, 9, 10 and 17 are primarily used to test the correct operation of the Bus and BCIU.

Mode Codes 4 through 7 are used to correct faults in Bus operation by manipulating the Bus interface.

Mode Codes 11 through 15 are used for Remote Terminals and are not of importance to BCIU operation.

Mode Codes of importance to remote BCIU operation are:

- 1 - Transmit Status Word. The status word is transmitted to the Master Processor BCIU.

- 9 - Initialize Terminal. This command initiates self test procedures so that at a later time the information may be transmitted via a Transmit BIT WORD (Mode Command 3) command.

- 10 - Transmit Last Command returns the contents of the Last Command Register. This command is used to determine whetner a command needs to be retransmitted.

- 18 - Master Function Interrupt. This is the Minor Cycle synchronization command. The data word following the command and indicating the minor cycle number is placed into the master Function Interrupt Register. The remote processor is then interrupted.

- 20 - Busy Override. This command sets the Busy continue Bit in the Remote BCIU Processor Control Register to OFF.

  The Remote Processor returns to normal operation.

- 21 - System Interrupt. This mode code causes the System Interrupt in the Remote Processor.

3.1.1.1.4 The Remote Terminal (RT)

3.1.1.1.4.1 Basic Characteristics - The Remote Terminal (RT) provides the interface between the IDAMST Multiplex System and the Aircraft Subsystems.

The RTs provide for bus communication with the IDAMST processors (as described in Section 3.1.1.1.2).

The subaddress field of each Transmit or Receive Command acts as a message identifier. The message is formatted by the RT for correct interface with the Interface Modules (IM) which relay (or accept from) the signals to the aircraft subsystems.

The RT performs all the error checking and setting of error and status bits.

3.1.1.1.4.2 RT Functions - The RT shall contain the registers, logic, decoders, buffers, comparators and control sequences required to perform the following functions:

a. Receive Command words from the Bus.

b. Detect Command words directed to this RT.

c. Receive Data words from the bus (one at a time) if directed to do so by the received Command word.

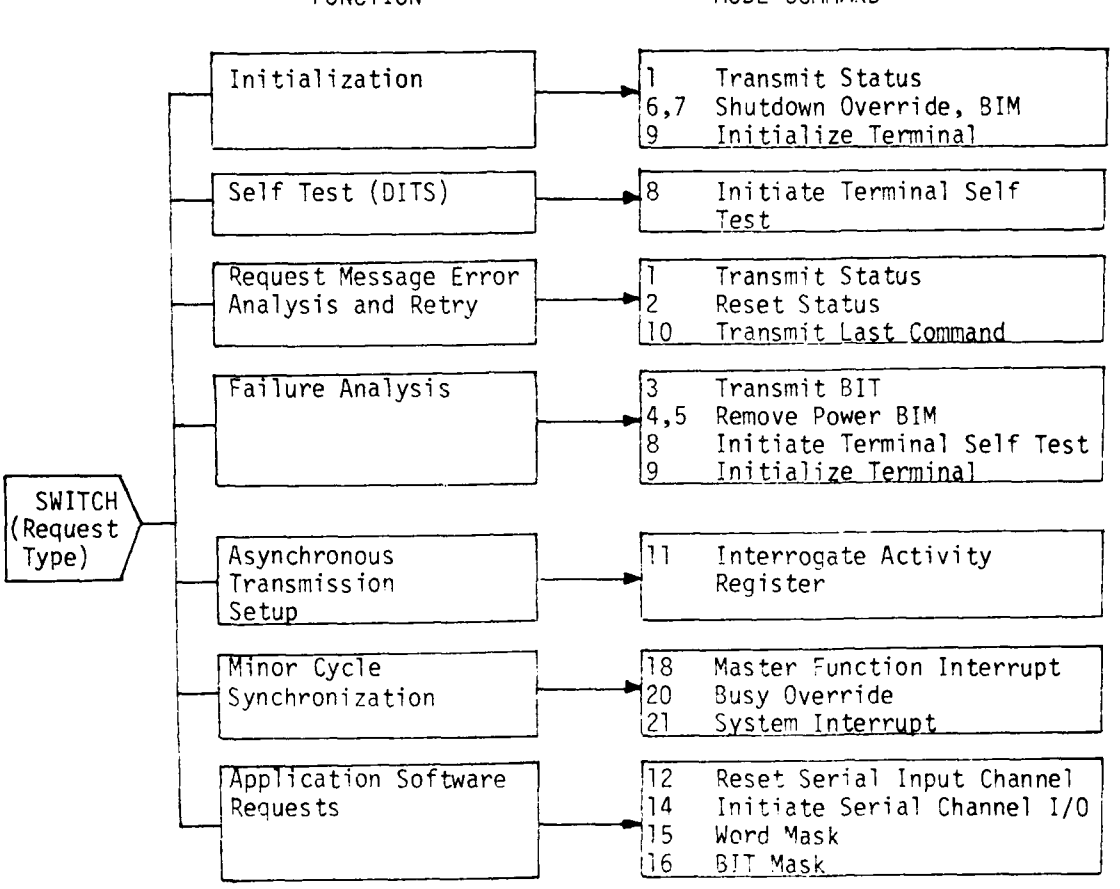


Figure 3.1.1.1.3.2.18-1 Software Interface with Mode Commands for BCIU and Remote Terminals

d. Transmit Data Words through the Bus to the data bus (one at a time) if directed to do so by the received Command Word.

e. Transmit Status Words through the Bus to the data bus as directed by the received Command Word.

f. Perform Mode Operations when and as directed by received Command Words.

g. Distribute received Data Words to the proper channels of the proper IMs.

h. Input Data words from the proper channels of the proper IMs for transmission to the data bus.

i. Maintain the Status Word and the Built-In-Test (BIT) Word of the RT by performing continuous and periodic self test functions within the RT.

j. Maintain an Activity Word and Error Word for monitoring status of serial digital IM's.

k. Maintain a Last Command Register for verification of command receipt in the event of an invalid response.

l. Perform Bit and Word Masking.

All data bus operations that the RT shall participate in shall be in the formats defined in the Multiplex Data Bus Specification, SA-301-200A.

3.1.1.1.4.2.1 Receive Data

The Command Word directs the RT to receive 1 to 32 data words with the number of data words involved being specified by the Word Count/Mode Code field. Each word of the received message shall be mapped to a subsystem output signal interface line as specified by the Subaddress/Mode and the Word Count/Mode Code fields.

3.1.1.1.4.2.2 Transmit Data

A Transmit Command prepares the RT to transfer from 1 to 32 Data Words, with the number of Data Words defined by the Word Count/Mode Code field of the Command Word. Each word of the message shall be sampled from the proper IM subsystem signal interface line in a predetermined order, using the Subaddress/Mode and Word Count/Mode Code fields to define which predefined order.

3.1.1.1.4.2.3 RT-Data Bus Mode Operations

When the Subaddress/Mode field in the Command Word is zeros, the Word Count/ Mode Code field is interpreted as a Mode Code. The Mode Codes are listed in Table 3.1.1.1.3.2-18-1, and an explanation of each one follows. Those Mode Codes that are not defined here may be used by the system designer. Any codes that are not used are declared to be invalid codes. The mode code of all zeros will not be used. If a terminal receives an invalid mode code, the terminal shall set the Invalid Command bit of the BIT word and, after the gap period, transmit its Status Word with the Message Error Bit set.

a. Transmit Status Word

Upon receipt of a command to transmit the Status Word, the terminal shall pause for the gap period and then transmit the Status Word. The format of the status word is as specified in SA-301-200A. Bit ten in the status code field shall indicate serial channel activity. Bit eleven in this field shall indicate serial channel parity error. The remaining seven bits of the status code field are undefined at this

time. The reception of the Transmit Status Mode Command shall not modify the contents of the Last Command Register, Status Word or BIT Word Register.

b. Reset Status Code Field

When the terminal receives the command for this mode operation, the terminal shall clear the nine bit status code field of the Status Word. After the field has been cleared, the Status Word shall be transmitted.

c. Transmit BIT Word

Upon receipt of a command to transmit BIT Word, the terminal shall pause for the gap period and then transmit the status word and then the BIT Word. A BIT Word shall be comprised of a sync waveform, a condition field, and a parity bit as shown in Figure 3.1.1.1.4.2.3-1 below. The reception of the Transmit BIT mode command shall not modify the contents of the Last Command Register.

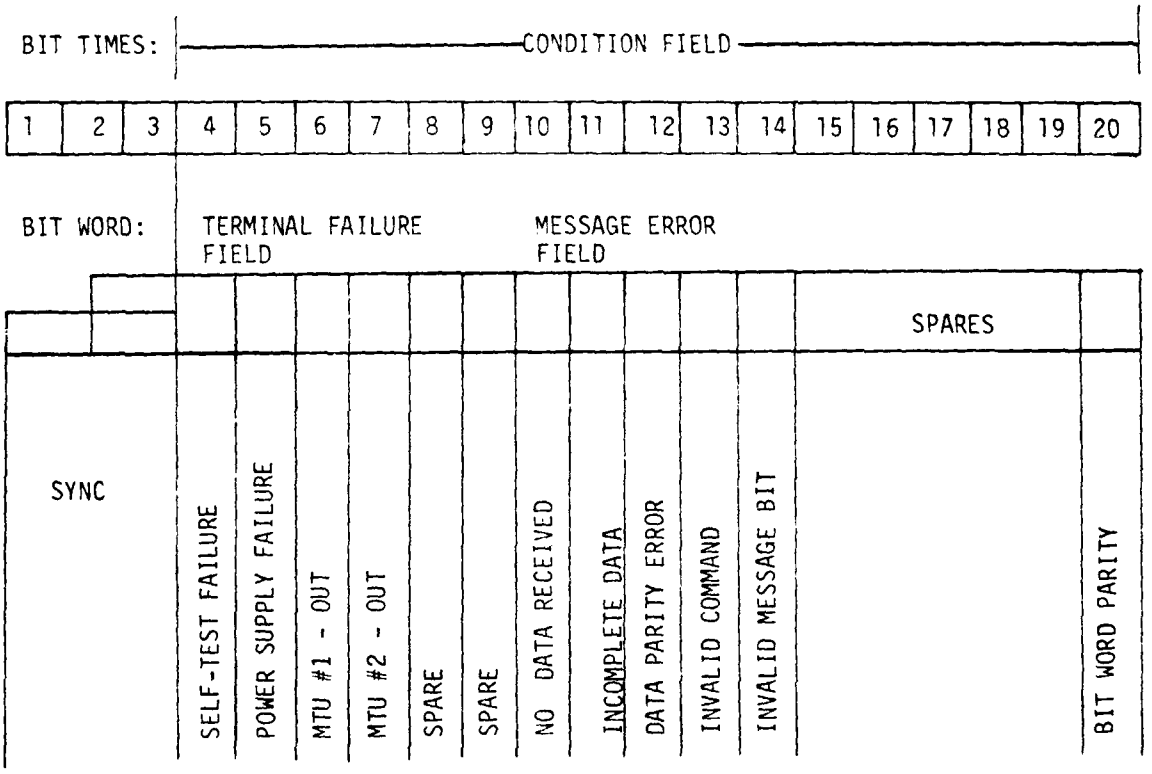


Figure 3.1.1.1.4.2.3-1 Composition of a BIT Word

The sixteen bits following the sync (bits 4-19) shall be utilized to indicate condition. If any particular bit is a logic "0" that condition is not true.

1) Terminal Failure Field - The first six bits (bits 4-9) of the Condition Field shall indicate failure conditions within the terminal, and shall be set and cleared by the conditions or the performance of self-test functions.

2) Message Error Field - The next four bits (bits 10-13) of the Condition Field shall indicate error conditions in the last data bus message that the Terminal participated in (excluding mode operations requesting status or BIT Word). This field is set to all logic "0's" when the Terminal begins operating on a new data bus message (again, excluding mode operations requesting status or BIT Word).

d. Remove Power - Bus Interface #1

When the terminal receives this command it will no longer listen for commands on the first wire of the Bus.

e. Remove Power - Bus Interface #2

Same as above, except that it applies to the second wire. Note that this means that not only does the terminal not listen for commands on that wire, but it does not send data or status words across that wire. Removing power from both Bus Interfaces removes the RT from the IDAMST system.

f. Shutdown Override - Bus Interface #1

When this command is received, the RT will again be able to receive and transmit across the first wire of the Bus.

g. Shutdown Override - Bus Interface #2

When this command is received, the RT will again be able to receive and transmit across the second wire of the Bus.

h. Initiate Terminal Self Test

When this command is received, the terminal executes its self test logic and reports any failures in the BIT Word.

i. Initialize Terminal

Upon receipt of this command, the terminal shall assume its initialization state, and after the gap period, transmit its Status Word.

j. Transmit Last Command

When the terminal receives this command, the terminal shall pause for the gap period and then transmit the Status Word followed by the con-

tents of the Last Command Register. The following mode operations will not be recorded in the register: Transmit Status Word, Transmit BIT Word, and Transmit Last Command.

k. Interrogate Activity Register

When the terminal receives this command, the terminal shall disable the activity bit in the status code field, transmit the Status Word, and then transmit the contents of the Activity Register. A one in the Activity Register shall indicate that a serial channel requests service.

l. Reset Serial Input Channel

The command word is followed by a data word that has the module and channel address. When the terminal receives this command, the terminal shall reset the serial channel Lockout Line, transmit the Status Word, and then reset the activity bit disable.

m. Interrogate Interface Module Error Register

When the terminal receives this command, the terminal shall disable the IM Error bit in the status code field, transmit the Status Word, and then transmit the contents of the IM Error Register. A one in the IM Error Register shall indicate a parity error on a serial channel.

n. Initiate Serial Channel Input/Output

The command word is followed by a data word that contains the Module and Channel address. When the terminal receives this command, the terminal shall reset the serial channel ERROR line, transmit the Status Word, reset the IM Error bit disable, and then input or output data to or from the subsystem.

o. Word Mask

Upon issuing this command, the BCIU shall follow with the transmission of two mask words from the Mode Data Register and then monitor the data bus for a status word response. A bit in the mask word set to logic 0 will mask the corresponding data word in the following Receive Message. The Word Mask Operation shall remain in effect for the next valid Receive Command received by the Remote Terminal. The word mask shall not be destroyed by Transmit or Mode command operations to the RT except the Initialize Terminal Mode Command shall cause the word mask to be cleared.

p. Bit Mask

Upon issuing this command, the BCIU shall then monitor the data bus for a status word response. The following receive message will consist of alternating Bit Mask and Data Words. A bit in the Bit Mask Word

set to logic 0 will indicate mask operation on the corresponding bit in the following data word. The maximum number of data words transferred shall be 16. The word count field of the Receive Command Word shall be the total of Bit Mask and Data Words. The Bit Mask Mode Command shall remain in effect until a Remote Terminal Receive Operation or an Initialize Terminal is received by the Remote Terminal.

q. No-op

Remote Mode - Upon receipt of this command the Remote Terminal shall store the command in the Last Command Register and respond with a status word.

Master Mode - Same as 3.2.1.1.1.3.10.

3.1.1.3 IDAMST Processor

The pertinent interfaces with the IDAMST Processor are described in Sections 3.1.1.2.1 through 3.1.1.2.5. The input/output organization is shown in Figure 3.1.1.2-1.

3.1.1.2.1 Vectored Interrupt System (See Figure 3.1.1.2.1-1 and Table 3.1.1.2.1-1

Sixteen levels of interrupt are provided for in hardware. The interrupt system shall have the capability of handling eight external and eight internal interrupt which alternate in priority rating with the highest being an internal interrupt. These sixteen individual interrupt lines shall feed sixteen flip flops which store the request. The four bits representing the interrupt number are used as four bits of an address. The format of the address is (00000000001 XXXX0). This address is used to fetch the address of a table where the condition status and instruction counter (IC) are to be stored. Upon interrupt, the present condition and instruction counter are stored at the fetched address. Further interrupts are disabled until the enable, ENBL, instruction is executed.

The new IC value is loaded from the location after the fetched address. The program may load and store the registers using LM and STM instructions. Return from an interrupt service routine is accomplished by the exchange status, EXS, instruction.

All interrupts, except power down, can be disabled by the disable, DSBL, instruction. Interrupts can be selectively masked by using the set interrupt control, SIC, instruction. The entire interrupt system will be automatically cleared upon power up.

An interrupt request may occur at any time but the interrupt processing must wait until an instruction is finished.

NOTE: The instructions move, MOV, move indirect, MOVI, and register shift instructions SLR, SAR, SCR, DSLR, DSAR, and DSCR can take many milliseconds. It is the programmer's responsibility to keep the count used by these instructions small enough to allow interrupts within a desired time.

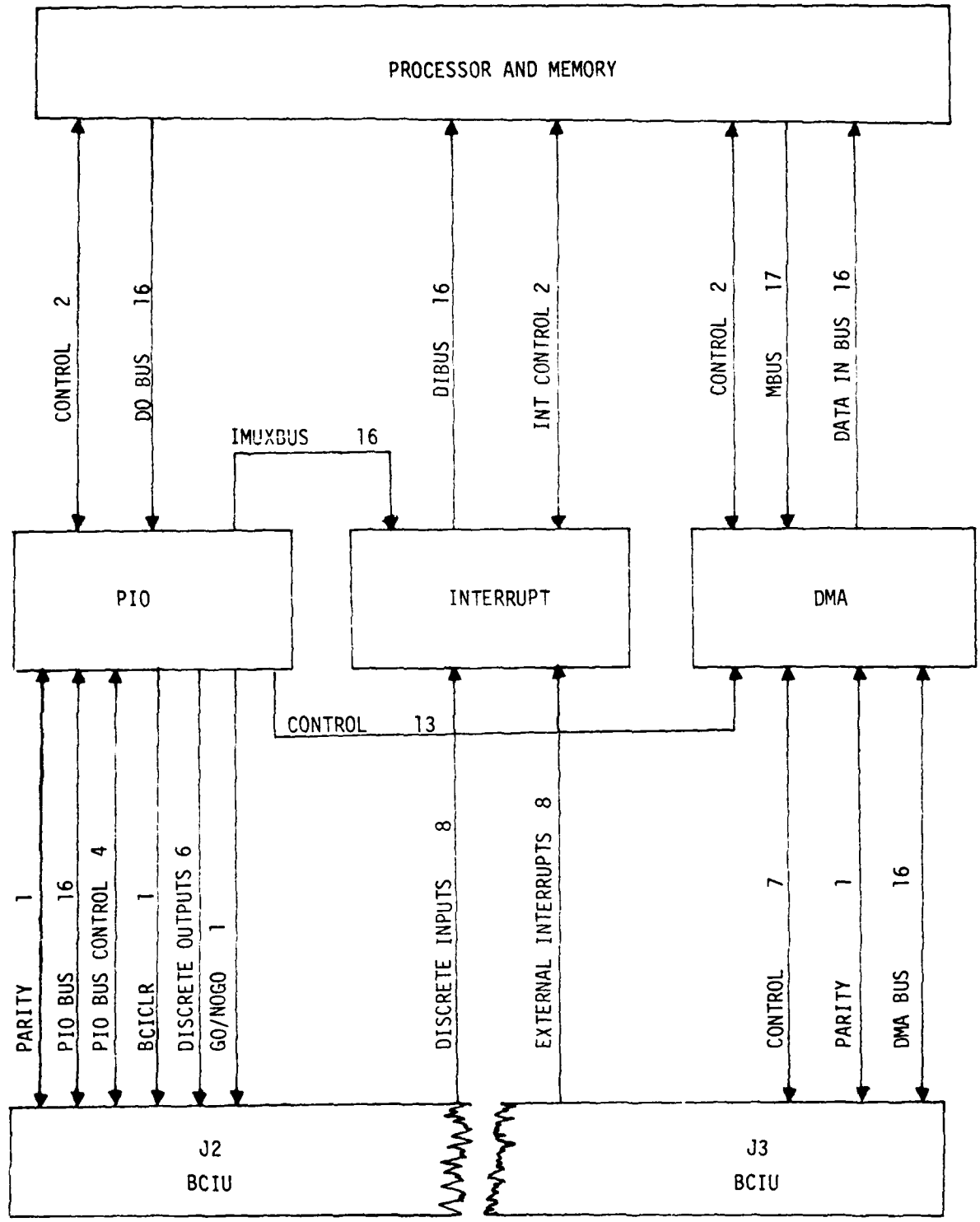
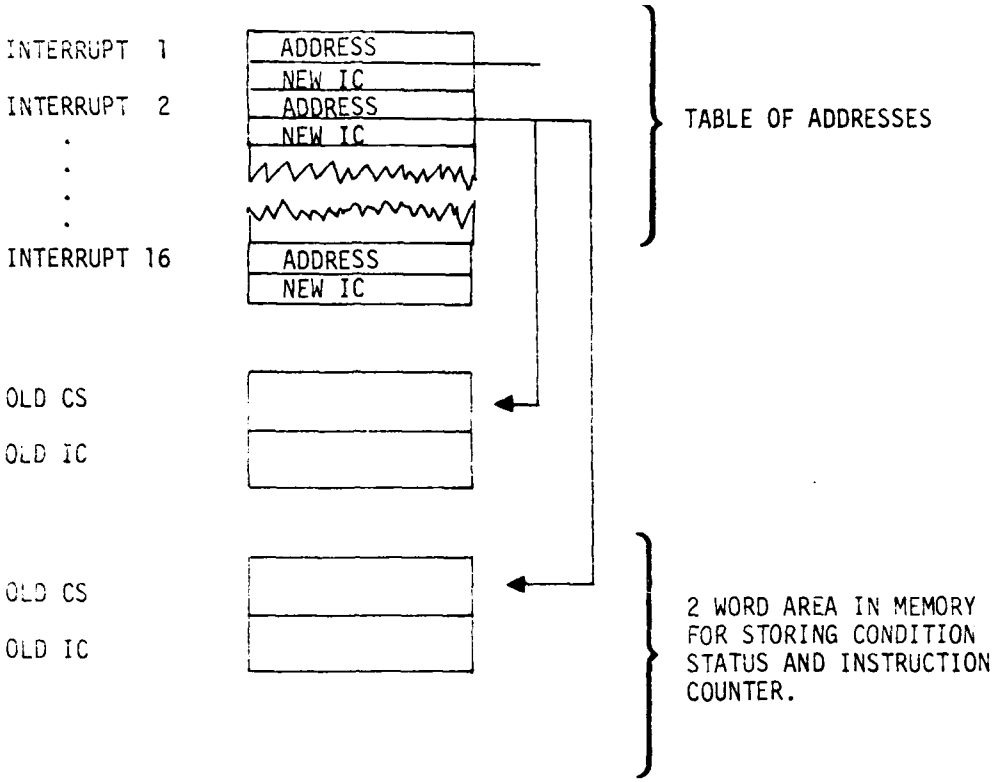


Figure 3.1.1.2-1 IDAMST Processor I/O Organization



The format in storage for the condition status upon interrupt, OLD CS, is:

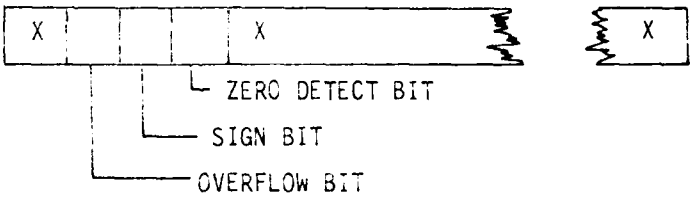
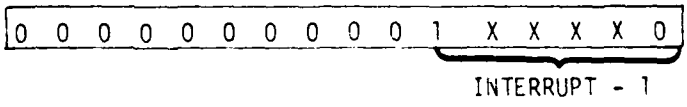


Figure 3.1.1.2.1-1. Interrupt Storage

INTERRUPT NUMBER	DESCRIPTION	EXTERNAL	TABLE POINTER	NEW IC
1	BCIU Interrupt #8	EXT	20	21
2	Spare	INT	22	23
3	BCIU Interrupt #7	EXT	24	25
4	Illegal Operation Code	INT	26	27
5	BCIU Interrupt #6	EXT	28	29
6	Boundary Alignment Error	INT	2A	2B
7	BCIU Interrupt #5	EXT	2C	2D
8	Interval Timer #2	INT	2E	2F
9	BCIU Interrupt #4	EXT	30	31
10	Interval Timer #1	INT	32	33
11	BCIU Interrupt #3	EXT	34	35
12	Processor Parity Error	INT	36	37
13	BCIU Interrupt #2	EXT	38	39
14	Processor Memory Protect	INT	3A	3B
15	BCIU Interrupt #1	EXT	3C	3D
16	Power Down	INT	3E	3F

NOTE: Interrupt 16 is highest priority and interrupt 1 is lowest priority. Interrupt 16 cannot be masked by the set interrupt control, SIC, instruction. The format of the table pointer address generated by the interrupt hardware is shown below.



NOTE: Interrupt 16, power down (a) cannot be masked, (b) is not disabled by the disable interrupt instruction, and (c) is not disabled upon occurrence of another interrupt. A minimum of 50 microseconds is available for processor operation from the time power loss is sensed. If primary power comes up immediately, there will be a 7.2 msec delay before power up.

Table 3.1.1.2.1-1. Interrupt Definition Table.

3.1.1.2.2 Special Memory Locations

Location (Hexadecimal)	Use
0	The first instruction executed upon power coming up starts at location zero.
20 through 3F	These are the locations for table pointers and new IC values for the vectored interrupts.
Instructions	Located where the programmer or project desires. Typically, this area is store protected.
Constants	Located where the programmer or project desires. Typically, this area is store protected.
Scratch Storage	Located where the programmer or project desires. Not store protected.

All of core memory is preserved upon power down.

3.1.1.2.3 Additional Storage

There is additional storage that is not a part of main memory. This storage has special uses and is not addressed as main memory. This storage is integrated circuit flip flops, registers, and random access storage. Upon power down and then power up, the contents of this storage may be unknown. The additional stora that software can influence are listed below:

16 registers of 16 bits each (A0 through A15)
4 bits of condition status (set to "zero" status upon power up)
64 bits of processor storage protect RAM
64 bits of DMA storage protect RAM
16 bits of interrupt masks
16 interrupt request flip flops
2 internal timers of 16 bits each
16-bit I/O holding register
16-bit instruction counter (cleared upon power up)

3.1.1.2.4 Discrete Inputs and Outputs and Interval Timers

Six Discrete Outputs shall be provided to the BCIU. Software shall output six bits to a holding register which feeds the differential drivers. The use of the outputs are TBS. Eight Discrete Inputs shall be accepted from the BCIU. Softwa shall input all eight bits simultaneously from a synchronizing register followi the receiver outputs. Two interval timers are provided within the processor. The timers are basically 16 bit counters. A 1 is added to the least significan bit of timer A every 10 microseconds and to timer B every 100 microseconds. Both timers can be loaded (by output instructions). As a *minimum*, timer B can be read by input instruction. An interrupt request is generated when the

timer increments from FFFF to 0000, if the proper mask register bit has been enabled. If the timers are not loaded, an interrupt request is generated after 65,536 counts.

3.1.1.2.5 Storage Write Protection

Upon power up, all memory is write protected. Typically, a program begins with instructions to load the storage protect RAM and then enables storage protect based on the RAM content. The first 64 bits of the RAM provide processor write protection. The next 64 bits provide DMA write protection. RAM addresses are used in input and output instructions to read and write the RAM. A 1 in the RAM provides write protection while a 0 in the RAM allows writing into memory. Each bit in the RAM applies to a 1024 word block of main memory. RAM address 0 applies to locations 0 through 1023. RAM address 1 applies to locations 1024 through 2047, ..., RAM address 63 applies to locations 64,512 through 65,535 for processor write protection. RAM address 64 applies to locations 0 through 1023, RAM address 65 applies to locations 1024 through 2047, ..., RAM address 127 applies to locations 64,512 through 65,535 for DAM write protection. Figure 3.1.1.2.5-1 shows a RAM which allows for expansion of access protection, which could encompass read or execute protection as well as write protection.

3.1.1.2.6 ROM Programs

The IDAMST processor will include programs in Read Only Memory (ROM), to load software upon system initialization and to perform self-test. The interface is (TBS).

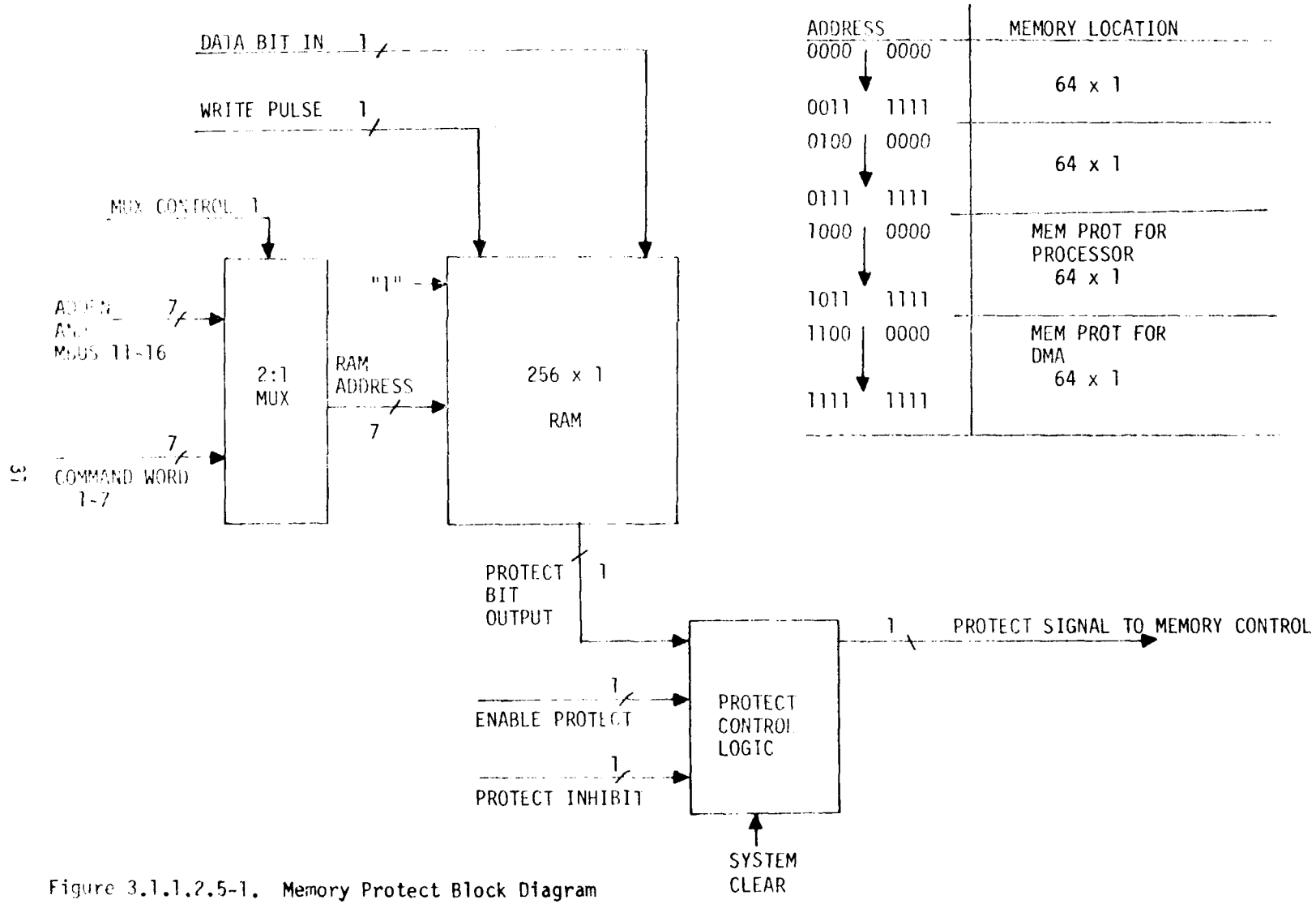


Figure 3.1.1.2.5-1. Memory Protect Block Diagram

3.1.1.3 Mass Memory

This section describes the functioning of the mass memory from which the IDAMST system may:

a. Read the programs from which to IPL (Initial Program Load) the system.
b. Read different program configurations to reconfigure the software during flight.
c. Write Digital Integrated Test System (DITS) results during the mission.

3.1.1.3.1 Mass Memory Commands

The tape drive shall receive the following commands through the control line as depicted in Figure 3.1.1.3.1-1.

3.1.1.3.1.1 Read Record - shall command the tape drive to read a variable length record (up to a limit such as 512 words). The record will be stored in a random access memory (RAM) (e.g. 512 words). Then the drive will halt.

3.1.1.3.1.2 Write Record - shall command the tape drive to write a fixed length record which is the size of the RAM. The record will be ascertained from the RAM. The drive will halt at the completion of the record.

3.1.1.3.1.3 Search Record - shall command the tape drive to search until the given record number has been established. The record number will be the initial word in the record. Once the record has been found, it shall be read into the buffer area.

3.1.1.3.1.4 Space Ahead/Back - shall command the tape drive to move to the beginning of other records relative to the present record.

3.1.1.3.1.5 Rewind - shall command the tape drive to rewind the tape to the beginning of tape marker.

3.1.1.3.1.6 Erase - shall command the tape drive to generate an interrecord gap and to write a predefined bit pattern (e.g. all 1's) at the normal data transfer rate.

3.1.1.3.1.7 Stop End Record - shall command the tape drive to stop at the end of the current record.

3.1.1.3.1.8 Halt - shall command the tape drive to halt operation.

3.1.1.3.2 Mass Memory - Remote Terminal Interface

The mass memory unit shall be attached to a Remote Terminal and be operated via the commands defined above. An intermediate multiple message buffer memory will be required. The buffer will allow several 32-word messages to be read into or written out of the buffer at the mass memory rate and to be accumulated or dispersed at the bus rate.

A buffer of 512 words would provide for an efficient packing of messages on a magnetic tape by having 16-32 word messages packed in a single record.

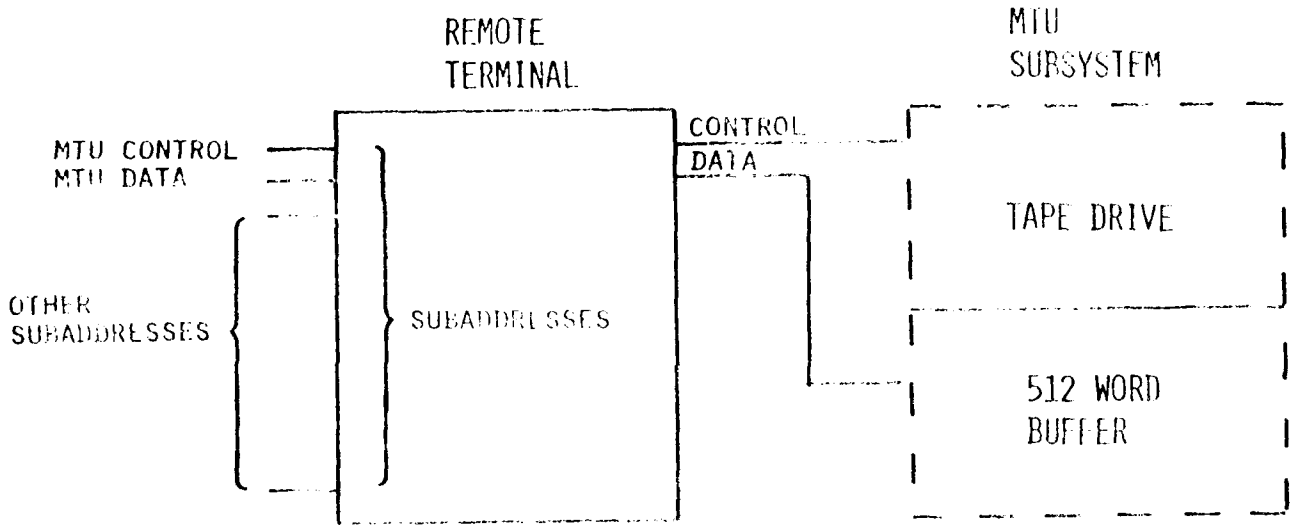


Figure 3.1.1.3.1-1
REMOTE TERMINAL/MTU INTERFACE

The remote terminal to buffer interface is also depicted in Figure 3.1.1.3.1-1. Two subaddresses provide the mechanism by which the 512 word buffer may be sequentially accessed in 32 word segments. One subaddress will be used for control and status information while the other subaddress is for data to and from the data buffer.

The status shall indicate whether the tape drive is at the beginning of the tape, end of the tape or at the black mark, is ready, busy, idle or running, the number of error retries that have occurred during that block transfer, and the command to which the drive is responding.

3.1.1.3.3 Tape Format

3.1.1.3.3.1 Tape Record Type Definition - An absolute binary tape consists of one or more records consisting of a 'record separator' character followed by a record number, followed by a single letter indicating the type of record which follows. Where applicable, the data follows the record type identifier character. The types of records which may be contained on an absolute binary tape are as follows:

Record Type Indicator	Record Type	Function
B	Binary Data	Actual binary data record
D	Header	Identification
E	Execution Address	Loaded into Processor Instruction Counter
G	End of Tape	Indicates physical end of tape.
H	End Load	Indicates end of data loading which may contain many files.

The following is a detailed description of each of the record types and how they are used. "R_S" is used to denote the "record separator" character.

3.1.1.3.3.2 Binary Data Record

R_S	RECORD NUMBER	B	LOAD ADDRESS	WORD COUNT	BINARY DATA	BINARY DATA	CHECKSUM

LOAD ADDRESS — 4 character code of the starting memory address where the data is to be sequentially loaded. For loads into non-contiguous memory locations, multiple Binary Data Records must be used. The most significant digit appears first.

WORD COUNT — 4 character code indicating the number of data words contained in the record. The most significant digit appears first.

DATA — 4 character code for each 16 bit word of the processor. The most significant digit appears first.

CHECKSUM — 4 character code which represents the modulo 2 sum (exclusive OR) of the binary data contained in the LOAD ADDRESS, WORD COUNT, and BINARY DATA FIELDS.

3.1.1.3.3.3 Header - Optional Record

RS	RECORD NUMBER	D	HEADER TEXT

HEADER TEXT — Character string which allows identification and other desired information to be associated with a tape or portion of it. The Header Text is terminated by the next 'record separator' character.

3.1.1.3.3.4 Execution Address-Optional Record

RS	RECORD NUMBER	E	EXECUTION ADDRESS

EXECUTION ADDRESS — 4 character code specifying the starting address to be loaded into the processor instruction counter. The most significant digit appears first.

3.1.1.3.3.5 End of Tape

RS	RECORD NUMBER	G	UNUSED TAPE	OPTICAL LEADER

Indicates physical end of tape. This is required because the clear leader at the beginning and end of a tape does not cause any signal to be generated from the tape cassette unit Receipt of this record before an End Load record signifies that an incomplete multiple tape load condition exists, and an appropriate message will be written on a display. Normal under these circumstances, the user would insert the next tape to be read and initiate a new tape read command, but this is not mandatory. The IDAMST system will require a single tape to load.

3.1.1.3.3.6 End Load

RS	RECORD NUMBER	F

Indicates the end of the set of records contained on a tape or tapes.

3.1.1.4 Processor Control Panel (PCP)

3.1.1.4.1 Physical Format

The physical format of the PCP is shown in Figure 3.1.1.4.1-1 below. Aircraft avionics power is supplied to the PCP when aircraft primary power or ground power is present.

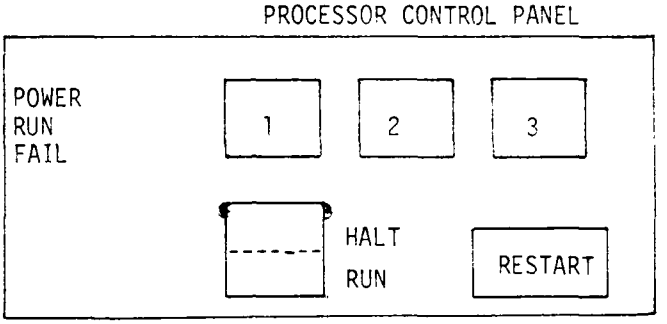


Figure 3.1.1.4.1-1 Physical Format of the PCP

3.1.1.4.2 RUN/HALT Switch

The RUN/HALT switch is a three pole single throw guarded toggle switch. When the RUN/HALT switch is moved to the RUN position and normal aircraft power available, each processor will be activated by a discrete signal. The RUN discrete is reset by processor action. When the switch is moved to the HALT position, each processor suspends operation.

3.1.1.4.3 RESTART Swtich

RESTART is a momentary operation pushbutton. Operation of the switch causes the 'RESTART' discrete signal to be sent to all mission processors in parallel. 'RESTART' discrete is reset by release of the RESTART button to normal position.

3.1.1.4.4 Processor Status Lights

One processor status light is provided for each IDAMST mission processor. These lights are capable of indicating three processor states via three distinct colors.

The processor status lamps assume nominal state one automatically when aircraft avionics power is presented to the PCP and the output of the mission processor, or associated memory or associated BCIU power supply voltage is not within tolerance (e.g. state one may indicate that a mission processor circuit breaker has disconnected power from the processor).

States two and three are set in the PCP by discrete signals from the mission processor. Absence of state two and three shall cause the processor status

lamp to revert to state one. State two indicates normal operation of the mission processor, memory and at least one channel of the redundant BCIU. State three indicates BIT failures of the mission processor, memory or all redundant channels of the BCIU.

3.1.2 Software Interfaces

The IDAMST Executive must interface with four other systems of software:

a. The Application Software
b. The Jovial J73/I Compiler
c. The Software Development and Verification System (SDVS)
d. PALEFAC

The Application Software, together with the RT's, constitute the totality of the entities upon which the Executive operates. In addition, most Executive actions are initiated by Real Time Pseudo-Statements within the Application Software.

The J73/I Compiler produces the object code for the Application Software and most of the object code of the Executive itself. The Executive must be cognizant of and compatible with the calling sequences used by the Compiler.

In the initial stages of software development, the Executive will run under the SDVS. The Executive must be compatible with the SDVS facilities, including the Statement Level Simulator and Functional Data Bus Simulator.

3.1.2.1 Application Software

The IDAMST Application Software consists of Tasks, Comsubs, Compool Blocks, and Events.

Tasks and Comsubs are processing modules, containing executable code and local data. Compool Blocks are data modules used for communication between Tasks. Events are boolean values used for control interactions between Tasks.

Tasks interact with the Executive through Real Time Pseudo-Declarations, and Real Time Pseudo-Statements.

3.1.2.1.1 Tasks

Tasks are the principal components of the IDAMST Application Software. Every Task is a J73/I Procedure, declared with no "data allocator", with no parameters or "function result".[1]

At any time, any Task in the IDAMST system has a "state". The possible states of a task are shown in Figure 3.1.2.1.1-1. Note that not all states are mutually exclusive; thus, a task which is "executing" is also dispatchable, active and invoked.

Immediately following system initialization, one Task, the Master Scheduler, is Invoked by the Executive, and all other Tasks are in Uninvoked state. There-

[1] See Jovial J73/I Computer Programming Manual, pp. 6-2 through 6-8.

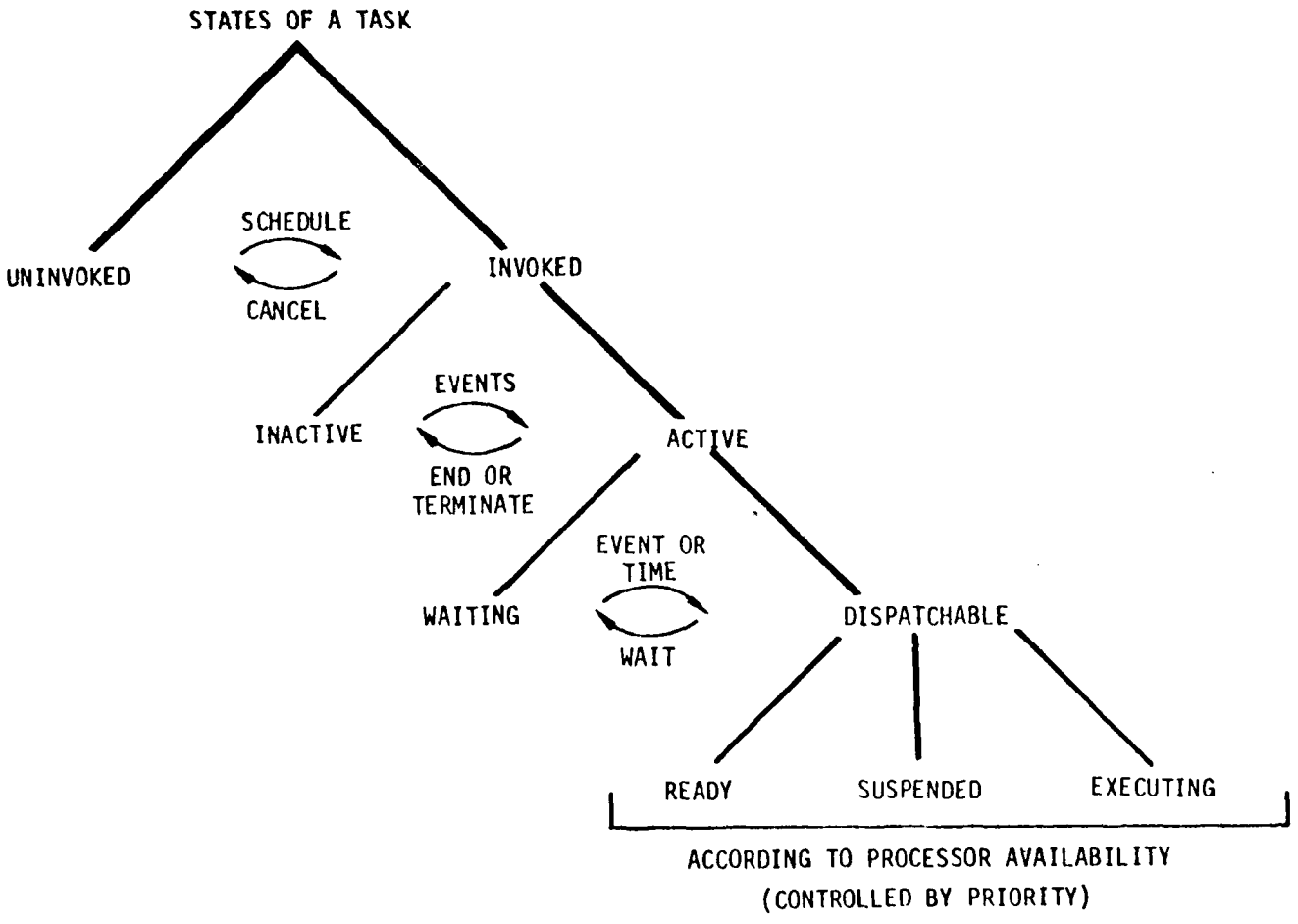


Figure 3.1.2.1.1-1 Task States and Control

after, Tasks can be put into Invoked state (Scheduled) or put into Uninvoked state (Cancelled) only by Real-Time Pseudo-Statements executed within other tasks.

Immediately after being Scheduled, a Task is Inactive; however, it has the potential to become Active, depending upon its Event Condition Set. The Event Condition Set is a collection of Conditions, each of which may be either "on" or "off". Each Condition has a "desired" value. When all the conditions in the Event Condition Set have their desired values, if the Task is Inactive, the Executive will put it into Active state. A Task may have a null Event Condition Set, in which case it can only be Inactive momentarily.

Each Condition in an Event Condition Set is associated with a set of Events. When any of these Events is set on, the Condition is set on; when any of these Events is set off, the Condition is set off. One Event may be associated with more than one Condition in an Event Condition Set. In addition, one Condition may be associated with a "Minor Cycle Event." These are Executive-generated Events which are set "on" at certain specified times (see 3.1.2.15) and are otherwise inaccessible to the Application Software. If a Condition is associated with a Minor Cycle Event, it may not be associated with any other Event.

A Condition may be either Latched or Unlatched. A Condition associated with a Minor Cycle Event must be Unlatched. The sole difference between a Latched and an Unlatched Condition is that upon the Scheduling or Activation of a Task, the Unlatched Conditions are set to the undesired value. Thus, a Task can only be Activated by an Unlatched Condition when the value of that condition is changed to the desired value subsequent to the last Scheduling or Activation of the Task. By contrast, Latched Conditions are changed only when one of their associated Events is changed. Therefore, a Task with only Latched Conditions in its Condition Set will be immediately Activated after it is Scheduled if all the Conditions were satisfied before the Schedule Statement.

A Task may return from Active to Inactive state from two causes: either because it completes execution, or because it is forcibly Terminated by another Task. In either case, immediately after it returns to Inactive state, the Event Condition Set is evaluated, and if all the Conditions have their desired values, the Task is immediately re-Activated.

When a Task is Activated, it is immediately put into Dispatchable state. If, at any point during its execution, a Task executes a Wait Statement, the Executive will place it into Wait state until the specified condition is satisfied, upon which the Task will again become Dispatchable.

All Dispatchable Tasks should theoretically be executed immediately. However, since there may be more than one Dispatchable Task at any time within any one of the Processors, Tasks are ordered by Priority to resolve possible conflicts. Whenever the Executive in any Processor is not called upon for immediate action it selects the highest Priority Dispatchable Task, and causes the Processor to execute it.

If a Task is Active but has not yet been executed, it is said to be Ready. If it has been in the process of execution, but has been interrupted by a higher priority Task, it is said to be Suspended. If it is executing, it is said to be Executing.

Any given Task may only be Scheduled by one Task, which is called its Controller. Two Tasks with a common Controller are said to be "siblings." The Task Scheduled by any Task are said to be its "sons". If a Task has no sons, it is said to have no "descendents"; otherwise, its descendents are its sons and all the descendents of its sons.

Only a Task's Controller may Cancel or Terminate it; however, when a Task is Cancelled or Terminated, all of its descendents are Cancelled or Terminated. If a Task attempts to Cancel or Terminate itself, it will Cancel or Terminate all of its descendents, but will leave its own state unchanged.

There are two types of tasks, Privileged and Nominal. Privileged mode shall be designated by a Privileged Mode bit in the Task Table B entry for the task. The Normal tasks have the Privileged Mode bit set to 0. When the Local Executive determines that a Privileged Mode Task is ready for activation, it will directly call the task. When a Privileged Mode Task executes, it will be in the privileged mode, and if it makes an Executive Service Request or is interrupted, the Local Executive shall return control directly to the Task. PALEFAC shall set the Privileged Mode bit in the Task Table B entry for the task and will place all of the Task Table entries for the Privileged Mode Tasks in a processor at the end of the Task Table for that processor, so that when the Dispatcher must search the Task Table the Privileged Mode Task Table entries will be examined first.

3.1.2.1.2 Comsubs

In addition to Tasks, the IDAMST Application Software may include another kind of processing module, known as the "Comsub". A Comsub is a Jovial J73/I based procedure declared external to any Tasks. A Comsub may be called from many Tasks; there is a copy of each Comsub in any processor containing a Task from which the Comsub may be called.

A Comsub communicates with a Task which calls it only through its parameters and/or function result. No Comsub may execute any Real-Time Pseudo-Statements; however, one Comsub may call another.

When a Task calls a Comsub, the Task is considered to be executing within the code of the Comsub. Thus, it is possible for one Task to be suspended within the code of a Comsub at the same time that another Task is executing within the same Comsub. In other words, a Comsub must be re-entrant. To implement this, every Task has a Comsub Local Storage Area assigned by PALEFAC for storage of local data by the Comsubs which it calls. At any time, there is a Comsub Stack Pointer which points to the area available for storage to the next called Comsub. This Comsub Stack Pointer is considered to be part of the process state of the Task, and is saved upon the occurrence of an Interrupt.

3.1.2.1.3 Compool Blocks.

All communication of data between Tasks or between Tasks and the external environment (RT's) is done by means of "Compool Blocks". No Task may

directly access a Compool Block unless a GLOBAL Copy is declared. Normally, a Task references a "Local Copy" which has size and attributes identical to the Compool Block. A Task may copy the Compool Block into its Local Copy by a READ Statement, or copy the Local Copy into the Compool Block by a WRITE or TRIGGER statement. From the point of view of the Application Software, READs, WRITEs, and TRIGGERs occur instantaneously, so a Compool Block can never be read when it has been partially updated by a WRITE. If a GLOBAL Copy has been declared, then the task in which the compool is declared GLOBAL Copy is allowed to access the GLOBAL Data Block directly, rather than using Executive Read and Write Requests into and out of local copies of GLOBAL Data blocks. The Executive Read and Write Requests will not actually move the data if the requesting task has declared the GLOBAL data block as a GLOBAL COPY rather than as a LOCAL COPY. The GLOBAL COPY provides for the same central control of table formats as the LOCAL COPY does.

Compool Blocks are divided into three classes: Input, Output, and Intertask. Input Compool Blocks can only be accessed by Tasks in a READ statement. Their values are determined by RT's. Output Compool Blocks can only be accessed by Tasks in a WRITE or TRIGGER statement; their values are "received" only by RT's. Intertask Compool Blocks are used solely for communication between Tasks.

Since a Compool Block may be accessed in more than one processor and also, possibly, in an RT, Compool Blocks may exist in multiple copies. Any processor in which a Compool Block is read has a Physical Copy of the Block; any RT which references the Block, or any processor which only WRITEs or TRIGGERs the Compool Block, is considered to have a Virtual Copy of the Block. To maintain consistency between the various copies of a Compool Block, the Executive must send Compool Update Messages across the Data Bus. Compool Blocks are further classified according to when these Update Messages are sent as: Synchronous, Asynchronous, and Critically Timed.

Synchronous Compool Blocks are updated from a single authoritative Copy, whether in a processor or an RT, at a specified rate and phase (see 3.1.2.1.5). All copies of an Asynchronous Compool Block are updated when any of those copies is changed, either by the hardware of an RT or by a WRITE statement within a processor. Critically Timed Compool Blocks are a special category used only for Output. They may only be TRIGGERed within a Task. A TRIGGER statement includes a "time to go". The Master Executive sends the Update to the appropriate RT at the specified time.

The various categories of Compool Blocks are shown in Table 3.1.2.1.3-1, along with the ways which they may be referenced in a Task.

The first word of each Physical Copy of a Compool Block is a "Minor Cycle Time Tag" which indicates the last time the Physical Copy was updated. See Figure 3.1.2.1.3-1 for the relationships of remote Terminals and Tasks to Compools.

	SYNCHRONOUS	ASYNCHRONOUS	CRITICALLY TIMED
INPUT	May be READ in in many Tasks.	May be READ in one Task	
OUTPUT	May be written in one Task.	May be written in many Tasks.	May be triggered in many Tasks.
INTERTASK	May be written in one Task, read in many Tasks.	May be written in many Tasks, read in many Tasks.	

Table 3.1.2.1.3-1 Categories of Compool Blocks

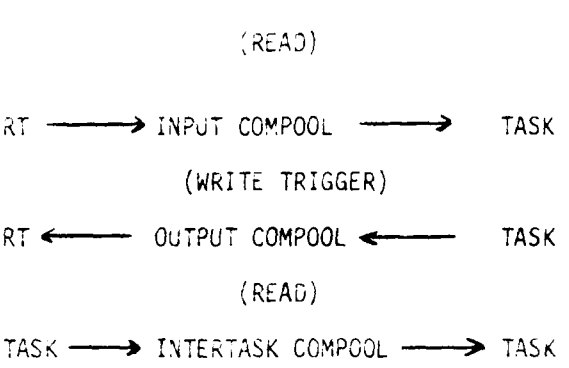


Figure 3.1.2.1.3-1 Relationship of Remote Terminals and Tasks to Compools

3.1.2.1.4 Events

Events are used for control communication between Tasks. An Event has two possible values: on and off. A Task may read the value of an Event, may WAIT on an Event (see 3.1.2.1.3) and an Event may appear in the Event Condition Set of a Task.

There are two general classes of Events: Application Events and System Events. Application Events are set on and off explicitly by Tasks. System Events are set on and off by the Executive upon certain occurrences. The initial value of all events is off.

System Events are further classified as:

a. Task Activation Events
b. Compool Update Events, and
c. Minor Cycle Events.

Any Task may have an associated Task Activation Event. Such an Event is set on when the Task is Activated and set off when the Task returns to Inactive or Uninvoked state. The Activation Event associated with a Task must have the same name as the Task.

Any asynchronous Task may have an associated Compool Update Event. Such an Event is set on when the Compool Block is updated, either by a Task or an RT. The Update Event associated with a Compool Block must have the same name as the Compool Block.

Minor Cycle Events are set on by the Executive according to specified rates and phases (see 3.1.2.1.5). They may only be referenced in Event Condition Sets.

3.1.2.1.6 Time

The Application Software may interact with time in two ways: it may reference absolute time, or it may specify that certain occurrences should

happen cyclically. Absolute time is measured in seconds from the initialization of the system. Cyclic time is maintained in terms of Minor Cycles and Major Frames.

A Minor Cycle is the shortest period of time at which a cyclic occurrence may be specified. A Major Frame is the longest period of time at which a cyclic occurrence may be specified. There are a fixed number of Minor Cycles to a Major Frame (currently 64), and each Major Frame has a fixed duration (currently one second). Every Minor Cycle is numbered in order of its occurrence within a Major Frame, starting with zero.

Cyclic occurrences are specified by period and phase. Period is the number of Minor Cycles between successive occurrences; phase is the Minor Cycle number of the first occurrence within any Major Frame. Clearly, $0 \leqslant \text{phase} \leqslant \text{period} \leqslant 64$.

In practice, Minor Cycles will not always occur exactly when they theoretically should, partly because the Data Bus may be overloaded in any given Minor Cycle. However, the Executive guarantees that these errors are not cumulative; it will always generate the next Minor Cycle as close as possible to the theoretical time, regardless of when the previous Minor Cycle occurred.

With one exception, the Minor Cycle is the finest granularity of time knowable with the IDAMST system. Thus, when a Task reads the absolute time, it receives the theoretical time of the last Minor Cycle. The sole exception to this rule is the Critically Timed Compool Block. When a Task TRIGGERs such a Compool Block, the Executive will attempt to send the Update Message to the RT at the precise time specified.

3.1.2.1.6 Real Time Pseudo-Declarations

Real Time Pseudo-Declarations are used to declare the real time entities referred to with a Task. There are four kinds of Real Time Pseudo-Declarations:

a. Task Declarations,
b. Event Declarations,
c. Compool Block Declarations, and
d. Comsub Declarations.

Task Declarations are used to declare Tasks referred to in Real-Time Pseudo Statements. They create a reference to the Task Table A entry for the appropriate Task.

Event Declarations are used to declare Events referred to in Real-Time Pseudo Statements. They create a reference to the Event Table entry for the appropriate Event. If the Event is a Compool Update or Task Activation Event, it must be declared as such in this Declaration.

Compool Block Declarations are used to declare any Compool Blocks referenced in READ, WRITE or TRIGGER statements. They do two things:

c. They create a reference to the Data Descriptor Block for the Compool Block, and

d. They access the Compool within which the Compool Block is declared, and from it create a declaration for the Local Copy of the Compool Block.

A Compool Block Declaration must indicate whether a Compool Block is read, written, updated (both read and written) or triggered within the Task.

Comsub Declarations are used to declare Comsubs called within the Task. They simply generate the appropriate REF PROC declaration.

3.1.2.1.7 Real Time Pseudo-Statements

The Application Software requests the services of the Executive through Real Time Pseudo-Statements. There are 11 kinds of Real Time Pseudo-Statements:

a. Schedule Statements
b. Cancel Statements
c. Terminate Statements
d. Wait Statements
e. Signal Statements
f. Read Statements
g. Write Statements
h. Trigger Statements
i. Event Statements
j. Task Condition Statements
k. Time Statements

Real Time Pseudo-Statements compile as calls to Executive routines, passing the appropriate information as parameters.

3.1.2.1.7.1 Schedule Statements

Schedule Statements are used by one Task to Schedule another Task. A Schedule Statement includes the following information:

a. The name of the Scheduled Task,
b. The priority of the Scheduled Task,
c. The latched Conditions, if any, in the Event Condition Set of the Task,
d. The unlatched Conditions, if any, in the Event Condition Set of the Task, and
e. The period and phase of a Minor Cycle Event, if any, in the Event Condition Set of the Task.

The latched and unlatched parts of the Condition Sets are defined by Event expressions. The syntax for Event expression is:

<Event expression> ::= <Condition> | <Condition> AND <Event expression>

<Condition> ::= <Event set> [NOT] <Event set>

<Event set> ::= <Event> | <Event set>

<Event> ::= <Event set> | <Event set>

Each <condition> in this expression corresponds to a Condition in the Event Condition Set. The presence of a NOT indicates that the desired value is off; the absence indicates that the desired value is on. The Events named in the <event set> are the Events associated with the Condition.

3.1.2.1.7.2 Cancel Statements

The Cancel Statement is used by one Task to put another Task into Uninvoked state. The Cancel Statement includes the name of the Task to be Cancelled. This Task must either be the Task within which the Statement is executed, or a son of that Task. If a son is cancelled all the descendents of the son are also cancelled automatically. If a Task attempts to Cancel itself, it will not affect its own state, but will Cancel all of its descendents. If a Task specifies itself in a Cancel Statement, it must be declared in a Task Declaration within itself.

3.1.2.1.7.3 Terminate Statements

The Terminate Statement functions identically to the Cancel Statement, except that it returns a task to the scheduled state.

3.1.2.1.7.4 Wait Statements

Wait Statements are used by Tasks to place themselves into Wait state pending certain occurrences. There are four kinds of Wait statements:

a. Absolute Time Waits
b. Relative Time Waits
c. Latched Waits
d. Unlatched Waits

An Absolute Time Wait places the Task into Wait state until a specified absolute time. If the specified time has already occurred, this statement is a No-Op.

A Relative Time Wait places the Task into Wait state for a specified period of time. If the specified period is non-positive, this statement is a No-Op.

A Latched Wait places the Task into Wait state until a specified Event reaches a specified "desired value." If the Event already has the desired value, this statment is a No-Op.

An Unlatched Wait places the Task into Wait state until the specified Event is changed to the specified value. This statement is never a No-Op.

3.1.2.1.7.5 Signal Statement

A Signal Statement sets a specified Event to a specified value.

3.1.2.1.7.6 Read Statement

A Read Statement copies the value of a specified Compool Block into the corresponding Local Copy. If the Compool Block is a GLOBAL Copy then no data transfer occurs.

3.1.2.1.7.8 Trigger Statement

A Trigger Statement requests the Executive to send the Local Copy of the specified Compool Block to the appropriate RT at a specified time. The specified time must be between two Minor Cycles and one Major Frame from the time the Trigger Statement is executed.

3.1.2.1.7.9 Event Statement

The Event Statement yields the value of the Event which has been passed as an argument. This Event must have been previously declared in an Event Declaration.

3.1.2.1.7.10 Task Condition Statement

The Task Condition Statement is applied to a Task. This function yields the value TRUE if the task INVOKED, FALSE if it is not.

3.1.2.1.7.11 Time Statement

The Time Statement returns the absolute time as a 31 bit signed integer signifying the elapsed time since system initialization.

3.1.2.1.8 Master Executive Interfaces

3.1.2.1.8.1 The Master Sequencer Interface

At the end of Master Initialization or Master Re-initialization the Master Executive schedules the Master Sequencer task. This task then schedules the other Applications Tasks.

3.1.2.1.8.2 Applications System Error Interface

Applications Software can detect error conditions and communicate the conditions to the Subsystem Status Monitor. The primary source of errors will be the Equips functions. These functions will determine any errant status with equipment and sensors and communicate the errors to the Subsystem Status Monitor.

The Subsystem Status Monitor records the error and gatners error statistics. If the last error was within too short a time or there were too many such errors, the Subsystem Status Monitor invokes the Configurator. The Configurator will cancel [illegible] if appropriate. If the errors are of such a magnitude to warrant reconfiguration, the Configurator can invoke the Reconfiguration function via the IO device function (see 3.2.1.3.10).

3.1.2.2 Jovial J73/I Compiler

Since certain portions of the Executive must be written in assembly language, the Executive must be cognizant of and compatible with the calling conventions used by the Jovial J73/I compiler. For use under Software Development and Verification System (SDVS) in the interpretive computer simulation (ICS) mode, the Executive must use the calling conventions specified for the IDAMST processor. In Statement Level Simulation (SLS) mode, the Executive must use the calling conventions specified for the DEC 10.

3.1.2.2.1 J73/I IDAMST Processor Run Time Conventions[1]

The procedure linkage convention is:

1. A parameter list is a series of parameter addresses, stored one per word. A function has an additional compiler generated parameter, described by the last entry in the parameter list, to receive the function value.

2. An @ procedure PP has DSIZE(PP) stored in the word immediately preceding the procedure entry point.

3. Register assignments are as follows:

   - 0 - Volatile, i.e., may be changed by called procedure.
   - 2 - Contains procedure return address (set by JS instruction)
   - E - @ space pointer, set to the called procedure's work space address when the called procedure is an @ procedure.
   - F - Parameter list pointer.

4. It is the responsibility of the called procedure to preserve the contents of registers 3 through E.

3.1.2.2.2 J73/I DEC-10 Run Time Conventions[2]

Procedure Linkage Conventions

The standard DEC-10 linkage convention is used by J73 object code. The convention used is as follows:

- R17 describes a linkage stack. The right half contains the address-1 of the next free stack word. The left half contains the complement of the number of words-1 unused on the stack.
- R16 is used as a parameter list pointer. A parameter list is a series of parameter addresses right justified and stored one per word. The

---

[1] From Appendix E of the Jovial J73/I Computer Programming Manual, Computer Sciences Corporation, October 1975.

[2] From Appendix D of the Jovial J73/I Computer Programming Manual, Computer Sciences Corporation, October 1975.

left half of each parameter list entry is zero. Preceding the parameter list is a parameter count word containing zero in the right half and the negative of the number of parameters in the left half.

- R0 is used as the function value return register for calls to FORTRAN functions. For J73 functions, an additional parameter (described by the last entry in the parameter list) is passed to receive the function result.

- Registers R0, R1, and R16 are considered to be volatile registers. That is, their contents need not be preserved by the called procedure. Registers R2 through R15 and R17 must be preserved by the called procedure.

- A call to a procedure P1 is done as follows:

```
MOVEI   R16,PL  ; GET PARAMETER LIST ADDRESS
PUSHJ   R17,P1  ; CALL P1
. . .           ; RETURN HERE
```

J73 Parameter Passing

J73 procedure parameters are passed using the standard DEC-10 parameter list convention. Each actual parameter is passed using a parameter list word as follows:

- Value input parameters - If a type conversion must be applied to the actual parameter or the actual parameter is not contained in storage exactly as the formal parameter (stored in consecutive full words) then the value of the actual parameter, converted if necessary, is assigned to the temp before the call. If this temp assignment is done, the address of the temp is passed in the parameter list. Otherwise, the address of the actual parameter is passed in the parameter list.

  The called procedure prologue (the initial code for the procedure) copies value input parameters using the address specified in the parameter list to the formal parameter.

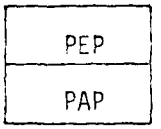
- Value output parameters - If a type conversion must be applied to the formal parameter before it can be assigned to the actual parameter or if the actual and formal parameter are not allocated to the same number of consecutive full words, the address of a temp which matches the formal parameter is passed in the parameter list. Otherwise, the actual parameter address is passed in the parameter list. The called procedure epilogue (the code executed immediately before procedure exit) copies each value output parameter using the address specified in the corresponding parameter list entry. Upon return from the procedure call, for each output value parameter for which a temp was passed, the calling procedure will copy the value contained in the temp to the corresponding actual parameter.

- Name parameters - For all name parameters, the address of the actual parameter is passed in the parameter list. The called procedure will

use this actual parameter address for all access to the formal parameter in the procedure.

- J73 function results - J73 function results are returned using a compiler generated value output parameter as the final parameter. The calling procedure supplies in the parameter list the address of a temp which will contain the function result value upon return.

- Parameter procedures - The address of a two word packet is passed in the parameter list for parameter purposes. The format of the packet is as follows:

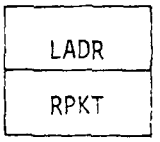


where

PEP is the procedure entry point address

PAP is the procedure @ space pointer. This value will be placed in R15 immediately before calling the procedure. It will only be used by a procedure which is internal to an @ procedure to locate the procedure's local storage.

Parameter labels - The address of a two word packet is passed in the parameter list for parameter labels. The format of the packet is as follows:



where

LADR is the address of the label.

RPKT is the address of a two word register save packet which contains values to load in registers R17 and R15 respecitvely. It is only valid to transfer to a label parameter if the procedure containing the label is currently active. The values for R17 and R15 are those established for the registers in the containing procedure prologue.

3.1.2.3 SDVS

To be useful in the development stages of the Application Software, the Executive must run under control of the Software Development and Verification System (SDVS). In Interpretive Computer Simulation mode, the Executive should be identical to

that used on the IDAMST processor. However, the Executive must also be able to interface with the Statement Level Simulator (SLS) and the Functional Data Bus Simulator (FDBS).

3.1.2.3.1 SLS

In SLS mode, those parts of the Executive which are written in assembly language must be written in DEC-10 assembly code. The SLS mode Executive must be compatible with the conventions used for the handling of simulated interrupts and with the calling conventions used on the DEC-10.

3.1.2.3.2 FDBS

When the IDAMST Executive is running with the Functional Data Bus Simulator (FDBS), the BCIU interface functions of the Local and Master Executives will be performed by the FDBS. The Executive must be able to interface with the conventions of the FDBS. These conventions are currently undefined.

3.1.2.4 PALEFAC

PALEFAC allocates and initializes the Executive Tables which drive the IDAMST Executive. These tables describe the attributes and interrelations of the various components of the IDAMST Application Software in machine-readable form. There are two categories of Executive Tables: Local Executive Tables, which are referenced by the Local Executive, and Master Executive Tables, which are referenced by the Master Executive.

3.1.2.4.1 Local Executive Tables

The Local Executive Tables are used for control of Tasks, Events, Compool Blocks and Comsubs. They also contain the information necessary for all I/O processing other than the control of the Master BCIU.

3.1.2.4.1.1 DMA Pointer Blocks

The Local Executive uses two 64-word blocks of pointers for DMA access by the BCIU. The first DMA Pointer Block is used on even numbered Minor Cycles; the second on odd. In each block some of the pointers are fixed and some are dynamically set up every time the DMA Pointer Block is used. PALEFAC should determine which pointers are to remain fixed and which should be maintained dynamically. If, for instance, fewer than 31 synchronous I/O blocks are referenced by a processor, then all the Pointers except the Asynchronous Pointer may remain fixed.

The format of each DMA Pointer Block is shown in Figure 3.1.2.4.1.1-1. The actual amount which is shown as fixed is, of course, hypothetical.

Each block must begin on an address divisible by 64.

3.1.2.4.1.2 Synchronous I/O Tables

The Synchronous I/O Tables are used by the Local Executive to determine which DMA Pointers to set up in the appropriate DMA Pointer Block on any given Minor Cycle. There are three tables used for this purpose:

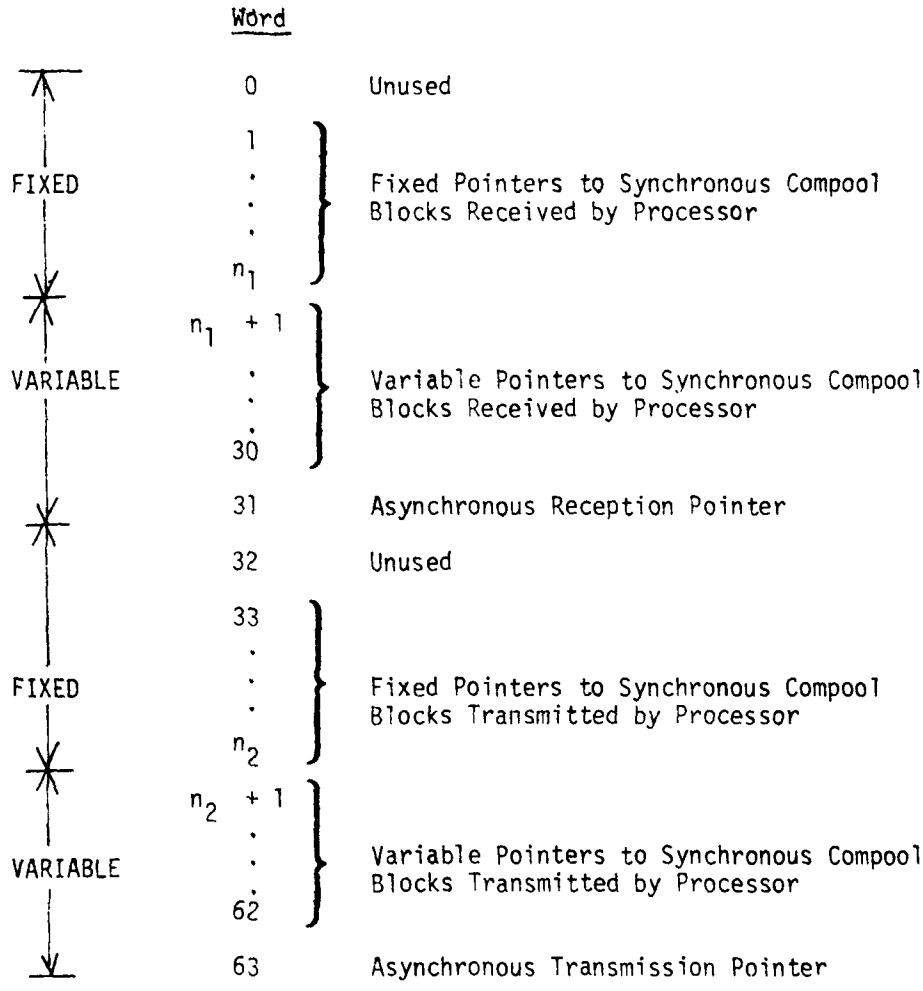


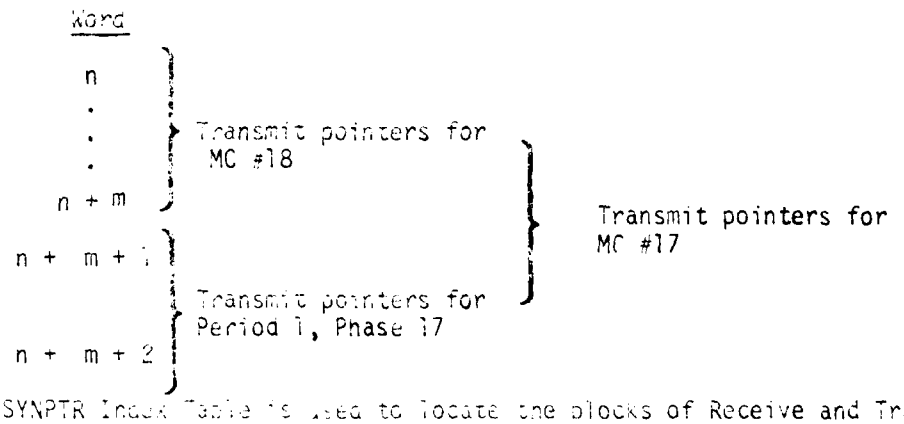
Figure 3.1.2.4.1.1-1 Format of a DMA Pointer Block

a. The Synchronous Pointer (SYNPTR) Table
b. The SYNPTR Index Table
c. The Pointer Block Descriptor

The SYNPTR Table contains two blocks of pointers for each Minor Cycle. One contains the addresses of all compool blocks received during the Minor Cycle, excluding the compool blocks whose addresses are fixed within the appropriate DMA Pointer Block. The other contains the addresses of all compool blocks transmitted during the Minor Cycle, excluding the addresses fixed within the appropriate DMA Pointer Block. Note that any one of these blocks of pointers may be null (if, for instance, all the DMA pointers are fixed within the DMA Pointer Blocks).

The blocks of pointers for two or more Minor Cycles may occupy the same physical location within the SYNPTR Table. For instance, if no blocks are received on Phases 1 or 33, Period 64 and no blocks are received on Phase 1, Period 33, then the data received on Minor Cycles 1 and 33 will be identical, so the blocks of Receive Pointers for these Minor Cycles need not be duplicated within the SYNPTR Table.

Furthermore, the block of pointers for one Minor Cycle may be wholly contained within the block of pointers for a different Minor Cycle. For instance, if no compool blocks are transmitted on Phase 18, Period 64, but two compool blocks are transmitted on Phase 17, Period 64, then the block of Transmit Pointers for Minor Cycle 17 will be:



The SYNPTR Index Table is used to locate the blocks of Receive and Transmit Pointers within SYNPTR for any Minor Cycle. It has one entry for each Minor Cycle Number. The information in each entry is:

Item	Description
1	Number of variable Synchronous Receive Pointers for this Minor Cycle.
2	Offset to First Receive Pointer in SYNPTR for this Minor Cycle.
3	Number of variable Synchronous Transmit Pointers for this Minor Cycle.
4	Offset to First Transmit Pointer in SYNPTR for this Minor Cycle.

The Pointer Block Descriptor is used by the Local Executive to determine which parts of the DMA Pointer Blocks are fixed and which are variable. The Pointer Block Descriptor contains four words:

Word	Description
0	Address of first variable Receive Pointer in Block 0
1	Address of first variable Transmit Pointer in Block 0
2	Address of first variable Receive Pointer in Block 1
3	Address of first variable Transmit Pointer in Block 1

Thus, for instance, the value of word 0 is LOC(DMA Pointer Block 0) + (number of fixed Receive Pointers).

3.1.2.4.1.3 Task Tables

To control the state of the tasks within its processor, the Local Executive uses two tables: Task Table A and Task Table B. Task Table B contains entries for each task resident in the processor. Task Table A contains entries for each resident task and for the controller and sons of such tasks, whether resident or not.

Task Table A is ordered according to the invocation tree, according to the following rules:

a. The controller of a task always precedes the task.
b. If Tasks A and B are siblings, if A precedes B, and A is not the controller of B, then every son of A precedes B.

The relative order of siblings is arbitrary. Note that this ordering extends across tasks in all processors and must be followed within the Task Table A of each processor, although no single Task Table A need contain entries for all tasks.

Whenever a task is referenced by the Application Software or by the Local Executive in another processor, the Task Table A entry for the task is referenced.

Task Table B is ordered according to priority, with the highest priority task first.

The Task Table B entry for any task is used internal to the Local Executive for referencing the task.

Table 3.1.2.4.1.3-1 represents the various items to be found in each entry of Task Table A. Within the Executive, entries in Task Table A are referenced by entry number, starting with one. Within the application software, a task BB is referenced by an REF TABLE T$BB, where LOC(T$BB) is the address of the Task Table A entry for BB.

Item	Description
1	Non-resident bit
2	Processor #
3	Pointer to Task Table entry
4	Non-resident bit for Controller
5	Processor # of Controller
6	Pointer to Task Table entry of Controller
7	Invoked/Uninvoked Bit
8	Number of Descendents

Table 3.1.2.4.1.3-1 Task Table A

Item #1 is on if the task is non-resident. If the task is non-resident, Item #2 is the processor where it resides; if the task is resident, Item #2 is zero. For resident tasks, Item #3 is an index to the entry for the task in Task Table B. For non-resident tasks, Item #3 is an index into Task Table A in the appropriate processor.

Items #4-#6 point to the controller in the same way that Items #1-#3 point to the task. These items may prove unnecessary in the ultimate implementation.

Item #7 is on if the task is invoked, otherwise it is off.

Item #8 is the total number of descendents of this task with entries in this processor's Task Table A. "Descendents" includes sons, grandsons, great-grandsons, etc.

Table 3.1.2.4.1.3-2 represents the various items to be found in Task Table B. Task Table B has entries only for resident tasks. Like Task Table A, it is referenced by entry number starting with one. Thus, if Item #3 of Task Table A for TASK has a value of N, the entry in Task Table B for TASK begins at LOC(Task Table B) + (N-1)(entry length for Task Table B).

Item	Description
1	Task Status (uninvoked/inactive/waiting/dispatchable)
2	Present Event Condition Set
3	Desired Event Condition Set
4	Latched/Unlatched Mask
5	Back Pointer for Wait Chain
6	Forward Pointer for Wait Chain
7	Time or Event Waited On
8	Starting Address of Task
9	Initial Value for COMSUB Stack Pointer
10	Save Area for COMSUB Stack Pointer
11	Save Area for Registers and PC
12	Restart Address
13	Pointer to Activation Event
14	Privileged Bit
15	Task Priority

Table 3.1.2.4.1.3-2 Task Table B

Item #1 is used to indicate the state of the task. The values for Item #1 are:

0 - uninvoked
1 - inactive
2 - waiting
3 - dispatchable

Item #2 is the Event Condition Set of the Task.

Item #3 is the Desired Event Condition Set.

Item #4 indicates which events in the Condition Set are unlatched; a "one" in any bit position means that the corresponding position in the Condition Set is for unlatched events.

Item #5-#7 are used to implement the WAIT statement.

The implementation of a timed response requires the establishment of a time queue. This time queue will be defined in real time by the Back Pointer and Forwards Pointer within each task in the queue. The beginning and end of a queue are marked by a null Back or Forwards Pointer. Since tasks are referenced starting with one, zero is an unambiguous "null" value.

If a task is waiting on time, Item #7 is time. Time, in the Executive, is measured in number of minor cycles from system initialization, maintained as a 31 bit unsigned integer.

If a task is waiting on an Event or the complement of an Event, the "queue" defined by the Back and Forwards Pointers will be used to identify all tasks waiting on the same condition. In this case, the high bit of Item #7 is on, and the low order bits point to the appropriate Event.

Item #8 is the address of the first executable statement in the task.

Item #9 is the address of the beginning of the Comsub Stack Area reserved for this task. See "Comsub Local Storage Area" for further information (see Section 3.1.2.4.1.8).

Item #10 is used to save the current value of the Comsub Stack Pointer when the task is dispatchable but not executing.

Item #11 is a save area for all information which must be remembered during an interrupt or a WAIT.

Item #12 is set, on task invocation, to the starting address of the task. Thereafter, it is set to the address following the most recent WAIT statement executed.

Item #13 is an offset from the beginning of the Event Table to the associated Activation Event for this task. If there is no such event, Item #13 is null

Item #14 is the flag to specify whether the task is to run as the highest priority task, and therefore be called directly from the dispatcher and return control without leaving privileged mode.

Item #15 is the priority assigned to the task and will be used to determine the relative order in which dispatchable tasks will run.

PALEFAC will supply the number of entires in Task Table B as a single word unsigned integer.

3.1.2.4.1.4 The Event Table

The Event Table contains an entry for each Event referenced with the processor. The information in each entry is shown in Table 3.1.2.4.1.4-1.

Item	Description	
1	Event Value	
2	Initialization Value, State 1	} Restart
3	Initialization Value, State 2	} Restart
4	Initialization Value, State 3	} Restart
5	Number of Non-Local Copies	
6	Number of Tasks Pointed to	
7	Pointer to Task Waiting on Event	
8	Pointer to Task Waiting on Complement	
9	Processor # of 1st Non-Local Copy	
10	Event Table Index of 1st Non-Local Copy	
.	.	
.	.	
.	.	
9	Processor # of nth Non-Local Copy	
10	Event Table Index of nth Non-Local Copy	
11	Bit Position in Task 1	
12	Task Table B Entry for Task 1	
.	.	
.	.	
.	.	
11	Bit Position in Task m	
12	Task Table B Entry for Task m	

Table 3.1.2.4.1.4-1 Event Table Entry

Item #1 is the value of the event.

Items #2-#4 are tentatively included as values for restarts. The subject of restarts requires further study.

Item #5 is the number of processors other than this one which reference the same event. This item indicates the number of items of types #9 and #10.

Item #6 is the number of tasks within this processor which include this event in their event condition sets. This item indicates the number of items of types #11 and #12.

Item #7 points to a task waiting on this event. If there is no such task, Item #7 is null.

Item #8 points to a task waiting on the complement of this event. If there is no such task, Item #8 is null.

Items #9 and #10 point to Event Table entries for this event in other processors. Item #9 is the number of the processor with such an entry. Item #10 is an offset from the beginning of the Event Table in that processor to the entry for this event.

Items #11 and #12 locate the position of this event within each event condition set including this event. Item #11 is the bit position within the event condition set, counting the leftmost bit as bit 0. Item #12 is the entry number of the Task Table B entry for the task with this event in its condition set.

The concept of "copy of an event" requires further elucidation. If an event is referenced by the same name in two different processors, the processors are considered to contain copies of that event except in the following cases:

a. Minor Cycle Events
b. Compool Update Events

These events signify occurrences within the Local Executive; therefore, if the "same" event is referenced in two processors, the two events are maintained independently, and are not considered to be copies of each other.

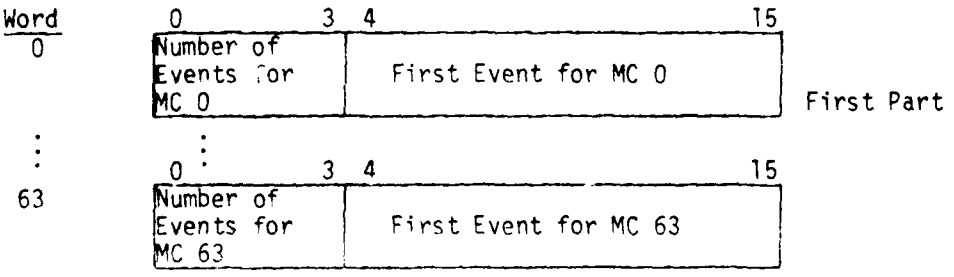
3.1.2.4.1.5 Minor Cycle Event Generation Table

Each Local Executive in IDAMST uses a Minor Cycle Event Generation Table (MC EGen Table) to determine which Minor Cycle Events to signal on any given Minor Cycle. The format of the MC EGen Table is modeled after that of the Synchronous I/O Tables.

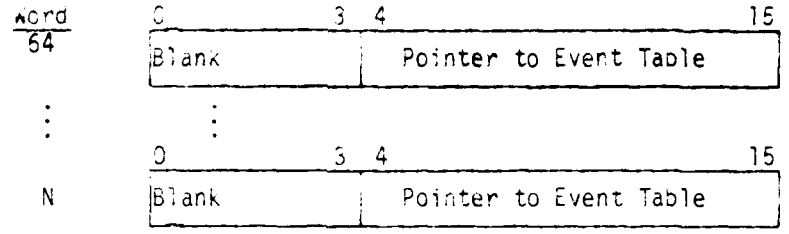
The MC EGen Table is divided into two parts: the first part contains two items for each Minor Cycle; a 4 bit count field and a 12 bit index field. The index field is an offset to the beginning of the second part of the Table; it points to the beginning of a list of Event Table pointers which point to the appropriate MC Event entries for the given Minor Cycle. If there are no MC Events for that Minor Cycle, the entire word will be zero.

Assuming 64 Minor Cycles per Major Frame, the format of the Table would be:

MC EGen Table



MC EGen Table (Continued)



Thus, for instance, to determine which MC Events to generate for Minor Cycle 13, the Local Executive will reference word 13 of the Table, and get a count C and an index N. Then words 64+N through 63+N+C of the Table are the pointers to the Event Table entries of the MC Events to be generated.

The actual arrangement of Event Table pointers within the second part of the Table may vary. Note that, in general, many distinct Minor Cycles will cause activation of an identical list of Minor Cycle Events, so the total number of distinct lists should be far fewer than 64.

3.1.2.4.1.c Compool Area

All communication between tasks and RTs, and all communication between tasks, except that done through the event and tasking mechanisms, must be accomplished by the use of compool blocks.

The contents of a compool block may be updated (altered) by the application software through the use of a WRITE or TRIGGER statement or by an RT. The contents of a compool block may be determined by the application software through the use of a READ statement, or they may be determined by an RT.

Any compool block potentially exists in multiple copies. Any processor which READs a compool block must have a physical copy of that block. Any RT which references a compool block, and any processor which WRITEs or TRIGGERs a compool block is considered to have a virtual copy of that block.

Compool blocks are classified according to the method used to maintain consister between the various copies as:

a. Synchronous
b. Asynchronous
c. Critically Timed

Synchronous compool blocks have a single authoritative copy, either in a process or an RT, from which all other copies are updated by the Master BCIU according to phase and period.

All the copies of an asynchronous compool block are updated whenever any of the is changed, either by an RT or by a processor. Any processor which has a physic copy of a compool block also potentially has an Update Event associated with the compool block which is set if when the copy of the compool block is changed, either by an internal task or a data bus message.

A critically timed compool block has one virtual copy in an RT, one physical copy in the Master Processor, and one virtual copy in the processor within which it is TRIGGERed, unless it is TRIGGERed by a task in the Master Processor. The copy in the Master is updated at the time the TRIGGER statement is executed; the virtual copy is updated at a "trigger time" which is specified in the TRIGGER statement.

All the physical copies of compool blocks existing within a processor are allocated by PALEFAC in a single, contiguous Compool Area. The format of each compool block is:

Synchronous and Asynchronous Compool Blocks

Word	Description
0	Minor Cycle Tag Word
1	Data
.	.
.	.
.	.
n	Data

3.1.2.4.1.7 DDB Areas

Every physical or virtual copy of a compool block within a processor has an associated Data Descriptor Block (DDB). All references to compool blocks except those within the Synchronous I/O Tables are done through DDBs.

There are three types of DDBs:

a. Synchronous DDBs
b. Asynchronous DDBs
c. Master Critical Timing DDBs

Synchronous DDBs are used for Synchronous Compool Blocks.

Asynchronous DDBs are used for all Asynchronous Compool Blocks and for any virtual copy of a Critical Timing Compool Block.

Master Critical Timing DDBs are used only for physical copies of Critically Timed Compool Blocks, i.e., copies existing within the Master Processor.

All Synchronous DDBs within a processor are contained in a single contiguous Synchronous DDB area. All Asynchronous and Master Critical Timing DDBs within a processor are contained in a single, contiguous Asynchronous DDB Area.

Synchronous DDBs

Item	Description
1	Sync/Async Bit
2	Number of Words in Compool Block
3	Address of Compool Block
4	Period - 1
5	Phase

Item #1 is on since the DDB is synchronous.

Item #3 is the starting address of the compool block, i.e., the address of the MC Tag Word.

Item #4 is the number of Minor Cycles between successive transmissions of this item. For instance, for compool blocks transmitted every MC, this value is zero.

Item #5 is the phase on which the compool block is transmitted. This may range from zero to the value of Item #4.

Asynchronous DDBs

Item	Description
1	Sync/Async Bit
2	Local Copy Bit
3	Update Event Bit
4	RT Transmit Bit
5	Number of Non-Local Physical Copies
6	Number of Words in Compool Block
(7)	Address of Local Copy
(8)	Offset to Update Event
(9)	Request Vector for RT TX
(10)	Offset to DDB of First Non-Local Physical Copy
(11)	Request Vector for First Non-Local Physical Copy

Items enclosed in parentheses exist only in some DDBs.

Item #1 is off because this is an Asynchronous DDB.

Item #2 is on if there is a physical copy of this compool block within this processor. This item indicates whether item #7 is present.

Item #3 is on if there is an update event for this compool block within the processor. This item indicates whether item #8 is present. Note that this item can be on only if item #2 is on.

Item #4 is on if there is a virtual copy of this compool block in an RT and this compool block is written within this processor. This item indicates whether item #9 is present.

Item #5 is the number of non-local physical copies of this compool block to be updated from this processor. It is zero if the compool block is not written within this processor. This item indicates the number of items of types #10 and #11.

Item #6 is the number of words in the compool block, including the MC Tag Word.

Item #7, if present, is the starting address of the local physical copy of the compool block.

Item #8, if present, is an offset from the beginning of the Event Table to the entry for the Update Event associated with this compool block.

Item #9, if present, is the Request Vector for updating the virtual copy of this compool block within an RT.

Items #10 if present, are offsets from the beginning of the DDB Area within each processor with a physical copy of this compool block to the DDB for that physical copy.

Items #11, if present, are the Request Vectors for sending updates for non-local physical copies of this compool block.

DDBs for Critically Timed Compool Blocks other than in the Master have Items #2-#4 off; Item #5-1, and Items #10 and #11 direct the update message to the Master Copy of the compool block.

Master Critical Timing DDBs

Item	Description	Bits
1	Master Critical Timing Code	8
2	Number of Words in Compool Block	8
3	Address of Compool Block	16
4	Triggered Bit	1
5	Request Vector	15
6	Forward Pointer to DDB	16
7	Trigger Time	16

Item #1 = octal 377 to identify this as a Master Critical Timing DDB.

Item #2 is the number of words in the compool block, excluding the MC Tag Word.

Item #3 is the starting address of the Master copy of the compool block.

Item #4 is on when the compool is waiting for the trigger time.

Item #5 is the Request Vector for sending the critically timed message to the appropriate RT.

Item #6 is an offset to the DDB of the next critically timed message waiting for a trigger time.

Item #7 is the time at which the message is to be sent to the RT.

3.1.2.4.1.8 Comsub Local Storage Area

Each Task must have a Stack Area for use by the Comsubs and Executive Services Routines which it calls. The length of each Stack Area must be:

a. At least long enough to accommodate the local storage for any Executive Service Routine, and
b. At least long enough to accommodate the local storage for the longest possible chain of Comsub calls initiated by the task.

All the Stack Areas for each task residing in a processor are allocated by PALEFAC within a single, contiguous Comsub Local Storage Area.

3.1.2.4.1.9 RT Reception Tables

The RT Reception Tables are used by the Local Executive to identify Asynchronous messages received from an RT. These messages are preceded by a word containing the address and subaddress from which the message was sent.

The RT address is used to index into the Terminal Originator ADdress Table (TOAD). The TOAD contains an entry for each of the 32 possible RT addresses. Each entry contains the following information:

Item	Description
1	Number of possible Asynchronous messages from this RT.
2	Entry in SNAKE for the first message from this RT

The Subaddress NAme Keys (SNAKE) contains an entry for each possible Asynchronous message from an RT. All messages from a single RT are contiguous within the SNAKE. Each entry in the SNAKE contains the following information:

Item	Description
1	Subaddress from which message was sent.
2	Offset into Async DDB Area to DDB for this message.

3.1.2.4.1.10 Number of Processors

PALEFAC will supply the total number of processors in the IDAMST as an unsigned integer.

3.1.2.4.2 Master Executive Tables

The Master Executive Tables are used by the Master Executive to control the operations of the Master CPU.

3.1.2.4.2.1 Master Request Update Table

Asynchronous [illegible] assignment to every possible processor to processor or [illegible] request block update message. The Request Vectors

are assigned starting at 16, and proceeding sequentially from there. Request Vectors 1 through 15 are reserved for interprocessor Service Requests.

The Master Executive uses the Master Request Decode Table to decode Request Vectors. Request Vector N corresponds to entry N of the Table, counting entries from one.

There are two types of entries in the Master Request Decode Table: one for transmissions involving no masking and one for transmissions requiring either bit or word masking. For messages with no masking, the entry is simply the Master Instruction Set to effect the transmission. For messages with bit or word masking, the information in the entry is:

Item	Description
1	Masking Flag = 0
2	Index into Master Instruction Supplement Table
3	Word Mask

Item #1 is zero, to indicate that this is not a Master Instruction Set.

Item #2 is an index into the Master Instruction Supplement Table (See 3.1.2.4.2.2).

Item #3 is the word mask, if the message requires word masking. If the message requires bit masking, this item is all ones.

3.1.2.4.2.2 Master Instruction Supplement Table

The Master Instruction Supplement Table (MIST) contains an entry for each Asynchronous message which requires bit or word masking. The entry is located by Item #2 in the Master Request Decode Table.

Each entry contains two Master Instruction Sets. The first is the Mode Command to indicate the proper kind of masking; the second is the command which performs the transmission.

3.1.2.4.2.3 Master Remote Terminal Request Tables

To decode asynchronous requests from Remote Terminals, the Master Executive uses two tables, the Remote Asynchronous Table (RAT) and the Master INstruction Keys (MINK).

RT asynchronous requests are identified by:

a. RT number, and
b. Bit position in the Activity Register

The RT number is used to index into the RAT. The RAT has 32 entries, one for each possible RT number. The format of the RAT is:

RAT

Item	Description
1	Number of possible requests for this RT
2	Index to first MINK entry for this RT

Item #1 is the total number of possible asynchronous requests which this RT may initiate.

Item #2 is an index to the first entry in the MINK for an asynchronous request which may be initiated by this RT.

The MINK contains an entry for each possible RT request. All the requests for ar given RT are contiguous within the table. The current format is:

MINK

Item	Description
1	Mask for Activity Register
2	Master Instruction Set

Item #1 is a mask for the Activity Register. It has one bit on, in the position which signifies this particular asynchronous request.

Item #2 is the Master Instruction Set to effect the necessary asynchronous transmission.

3.1.2.4.2.4 Master Synchronous I/O Tables

The Master Synchronous I/O Tables are used by the Master Executive to perform the appropriate synchronous Bus transactions every Minor Cycle. Two tables are used for this purpose:

a. The Synchronous Instruction List
b. The Instruction List Pointer Table

The Synchronous Instruction List contains one block of Instructions for each phase and period for which there are synchronous Bus transactions. Thus, it contains up to 2 (MC)-1 blocks, where MC is the number of Minor Cycles per Major Frame. Each block contains one Master Instruction Set for each synchronous Bus transmission to be performed for that phase and period, plus a Link Instruct Pair. The Link Instruction Pairs are dynamically set by the Master Executive to point to the next block of instructions to be performed during the current Minor Cycle. The Link Instruction of the last instruction block is set to a halt Instruction Pair within the Executive.

If any of the synchronous transactions require bit masking, the Master Instruction Set which effects the transmission will be preceded by one to send the proper Mode Command. If a transaction requires word masking, the transmission Instruction Set will be preceded by two Instruction Pairs. The first will be a No-Op Instruction with the interrupt bit set. The second will be the appropriate Mode Command. The word mask will be contained in the second word of the No-Op Instruction.

Synchronous Instruction List

Word		
0	Master Instruction Set	block for phase = 0, period = 1
2	Master Instruction Set	
.	.	
.	.	
.	.	
N	Link Instruction Pair	
.	.	
.	.	
.	.	
M	Master Instruction Set	block for phase = 63, period = 64
.	.	
.	.	
.	.	
L	Link Instruction Pair	(if it exists)

The Instruction List Pointer Table is used by the Master Executive to set the appropriate Link Instruction Pairs every Minor Cycle. It contains 2(MC)-1 entries, where MC is the number of Minor Cycles per Major Frame. There is one entry for each phase and period. The entries are arranged in ascending sequence first by phase and then by period, so that the entry for phase = PHASE and period = PERIOD is entry number (PHASE + PERIOD), where the entries are counted starting with one.

Instruction List Pointer Table

Item	Description
1	First Word in Instruction Block
2	Last Word in Instruction Block

Items #1 and #2 are the absolute addresses of the last words of the instruction block for the period and phase corresponding to the entry in the Table. Note that item #2 points to the second word of a Link Instruction Set. If there is no instruction block for this phase and period, items #1 and #2 are zero.

3.1.3 IDAMST Executive Functional Description

Every processor in the IDAMST system contains Executive software. Executive software consists of two parts: A Master Executive and a Local Executive. In general, the Master Executive provides system-wide services, such as data bus management and system control functions, while the Local Executive provides services to the Tasks located in a processor.

Each processor contains either a Local Executive or a Local and a Master Executive. Specifically, the Master processor and the Monitor (backup Master) processor contain a Master Executive, while the other processors do not.

The code of all Local Executives is identical. Each Local Executive can operate in either Remote or Master Mode. In Master Mode, the Local Executive supports the functions of a Master Executive, while in Remote Mode the Local Executive is not cognizant of the Master Executive, even it is present in the same processor.

The Local Executive in the Master Processor always operates in Master Mode; the Local Executive in the Monitor Processor operates in Remote Mode except in the case of Reconfiguration. The Local Executives in the Remote processors always operate in Remote Mode.

Normal flow of control and data for Synchronous and Asynchronous processing in Remote and Master Modes are shown in Figures 3.1.3-1 through 3.1.3-5.

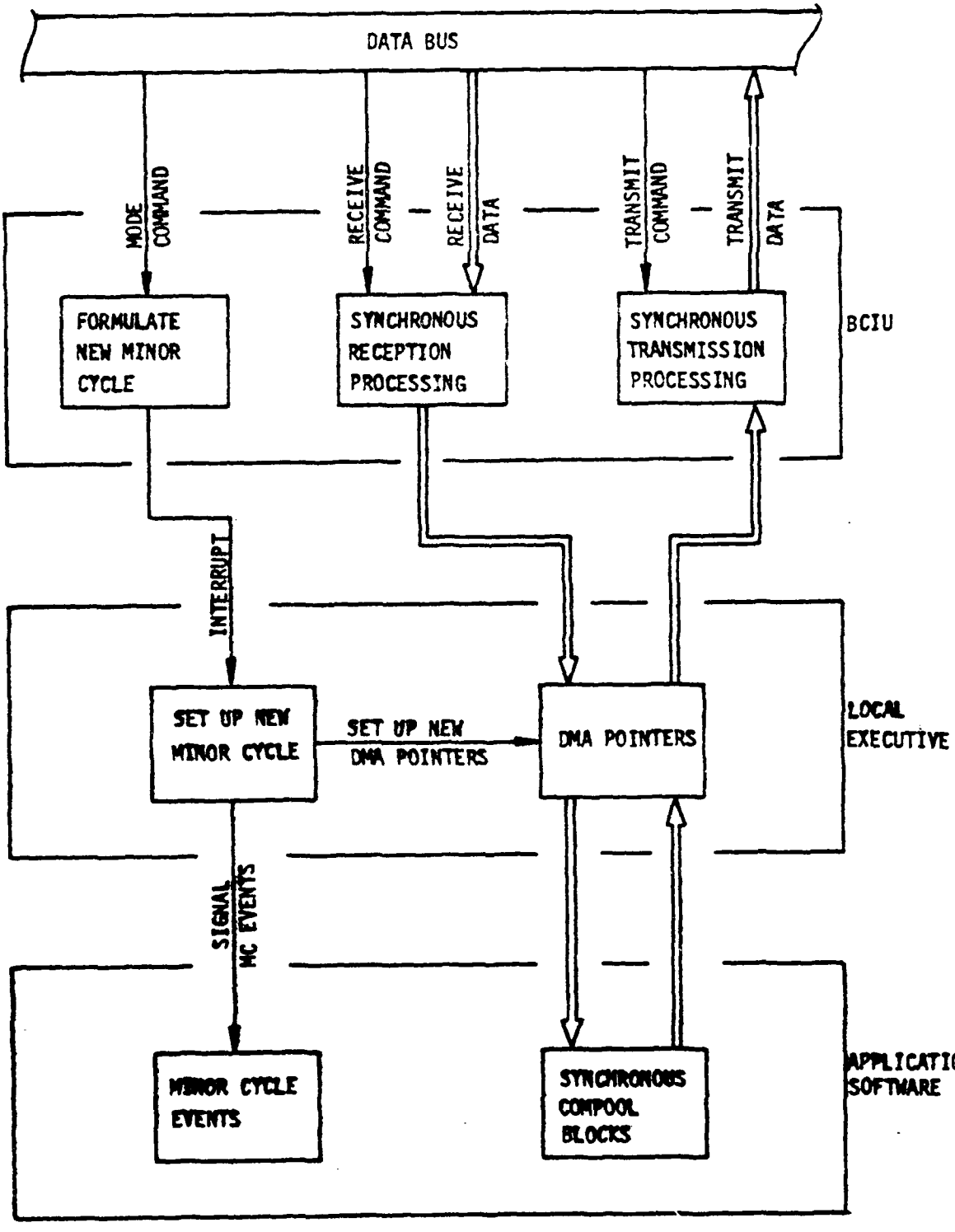


Figure 3.1.3-1 Synchronous Processing in Remote Mode

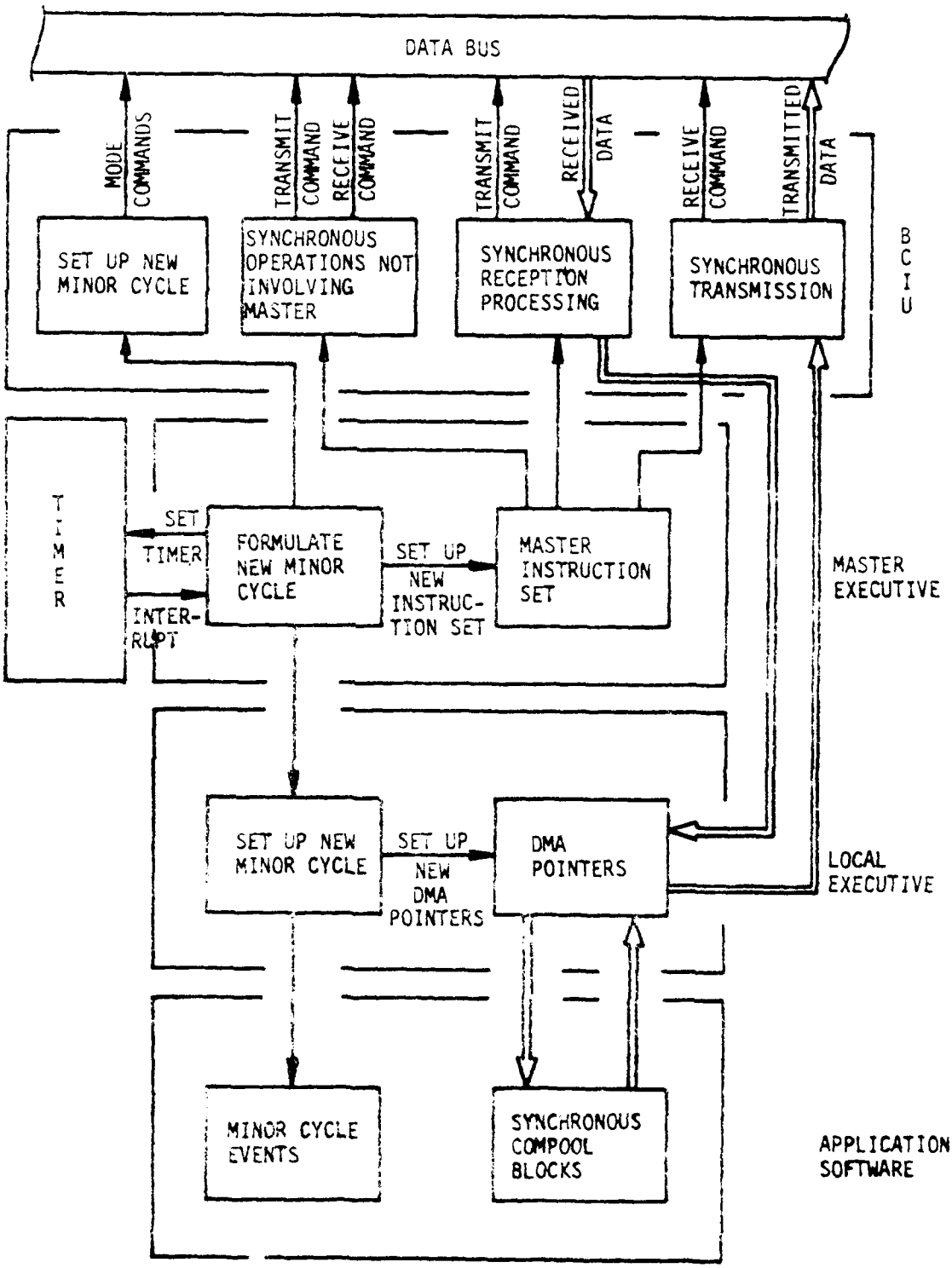


FIGURE 3.7.3-2. SYNCHRONOUS PROCESSING IN MASTER MODE

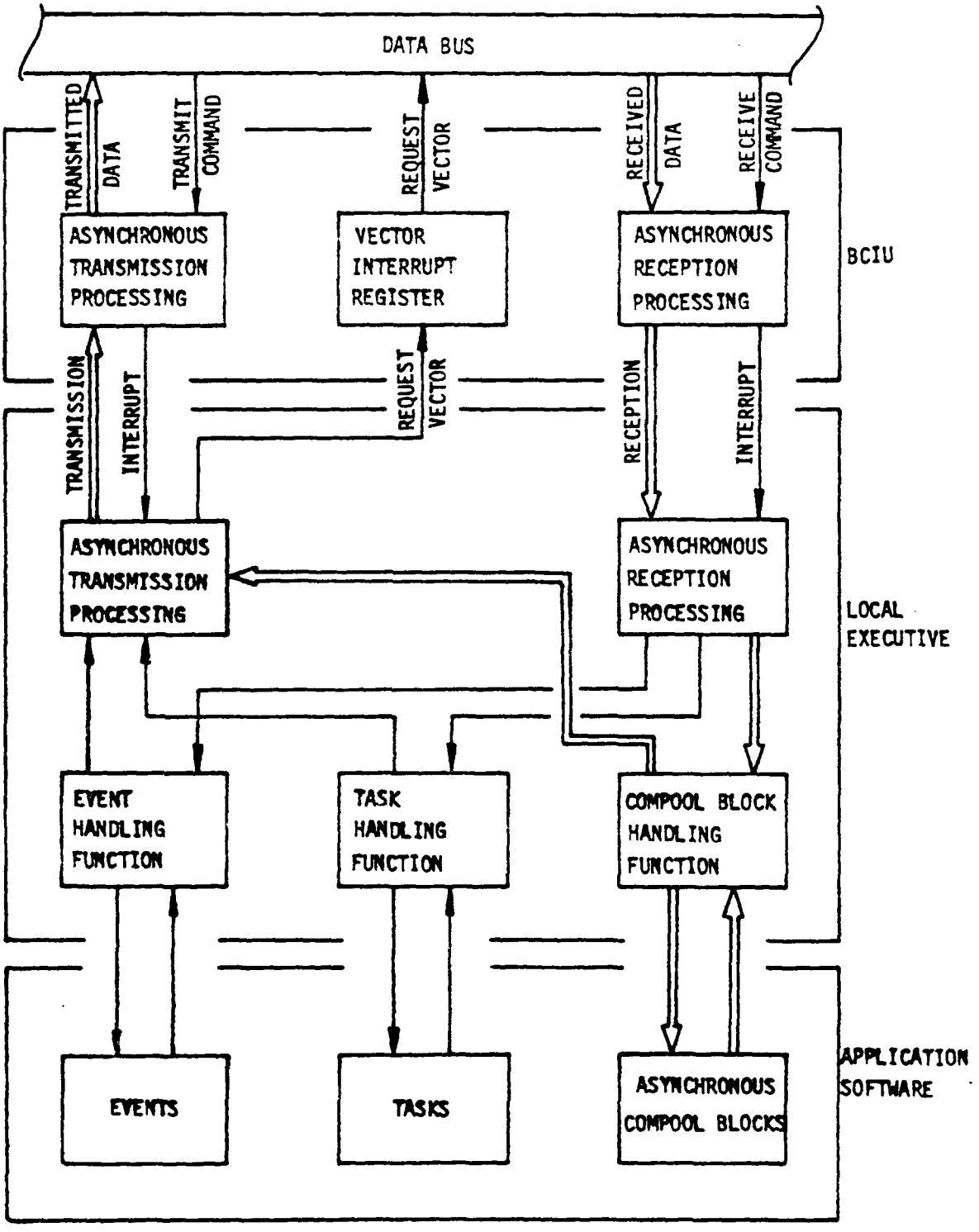


Figure 3.1.3-3 Asynchronous Processing in Remote Mode

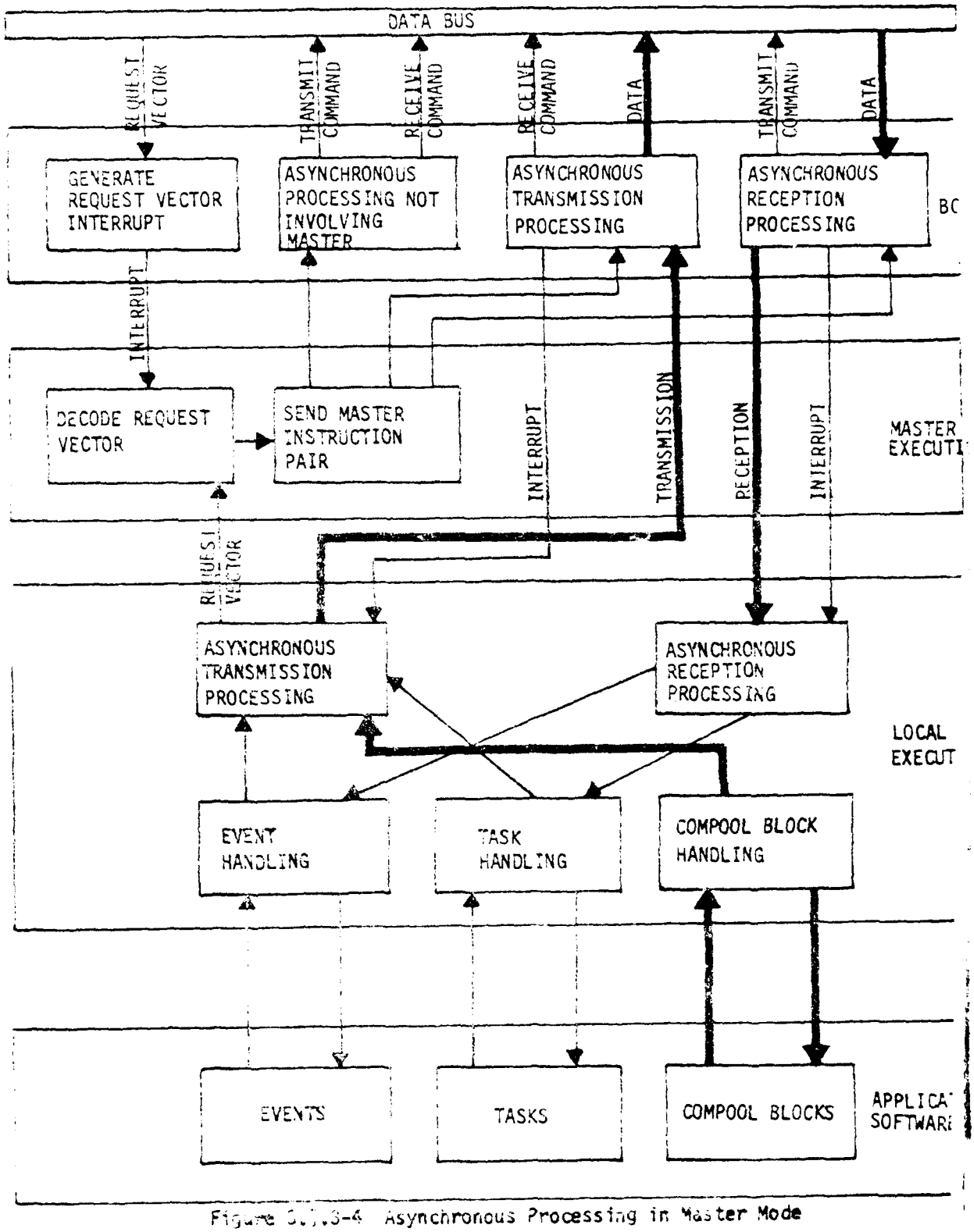


Figure 3.1.3-4 Asynchronous Processing in Master Mode

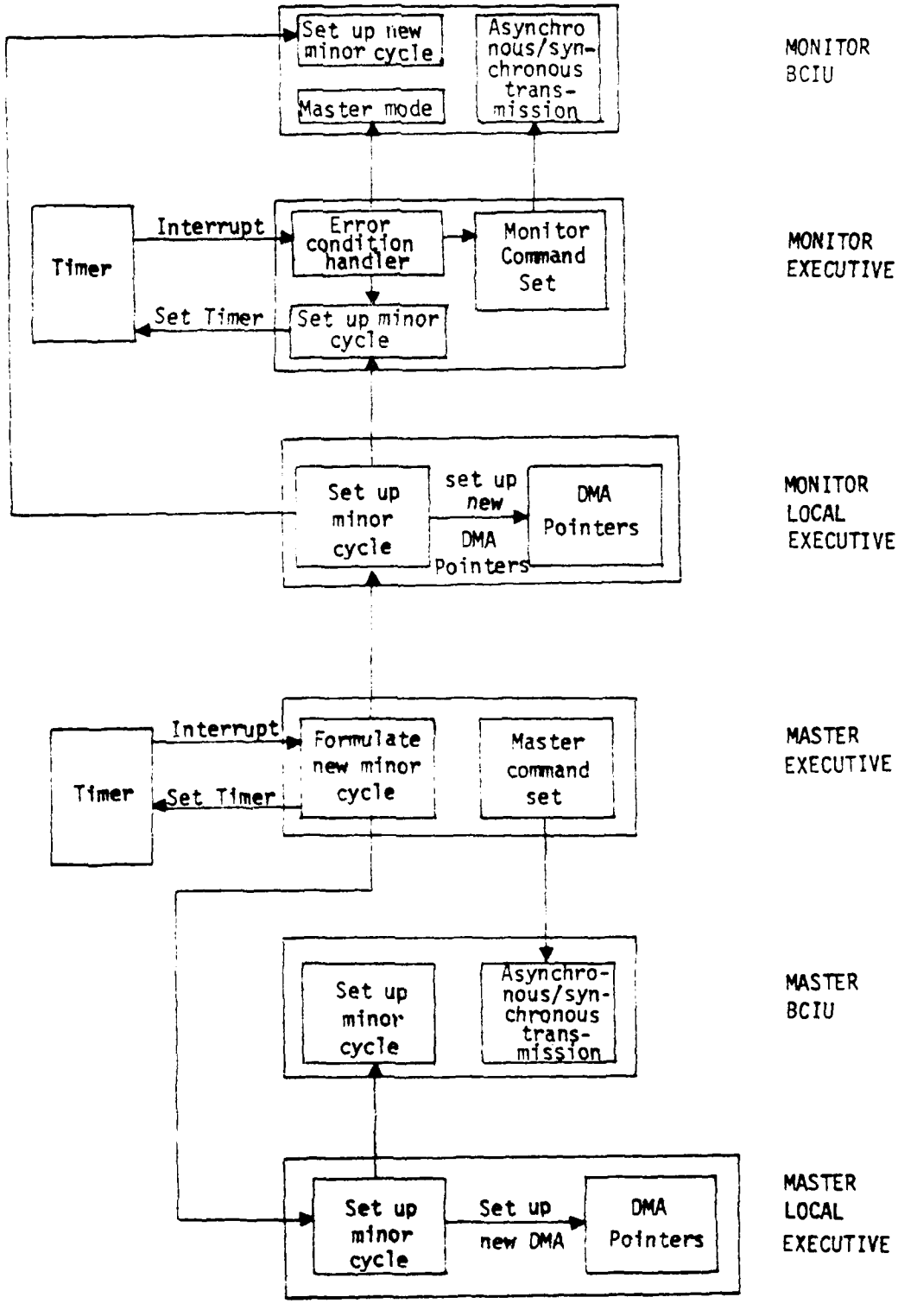


Figure 3.1.3-5 Master-Monitor-Local Executive Processing

3.1.3.1 Local Executive

The IDAMST Local Executive may be entered in two ways: through the execution of a Real Time Pseudo-Statement in the application software, or through an External Service Request from the BCIU or the Master Executive. To avoid re-entrancy and maintain proper sequencing, the processor may operate in any one of the following modes:

a. Normal Mode
b. Privileged Mode
c. Uninterruptable Mode

Normal Mode is the mode of operation for Application Software. In this mode, any External Service Request received by the BCIU will cause immediate entry to the Executive to service that Request.

Privileged Mode is the normal mode of operation for Executive routines. It is marked by the fact that the Privileged Mode Flag is on. In this mode, any External Service Request received by the BCIU will be queued for servicing at a later date.

Uninterruptable Mode is used for the immediate servicing of interrupts, and for certain critical operations within the Executive. It is marked by the fact that all interrupts are disabled. In this mode, External Service Requests are not recognized.

For purposes of modularity, the Executive is divided into four sections.

a. The Hardware Interface Control Function
b. The Application Interface Function
c. The Local Executive Proper
d. The Local Executive Initialization and Recovery Function.

The Hardware Interface Control Function responds to interrupts, and in general supports the interface with the BCIU. In Master Mode, the Master Executive runs under the hardware Interface Control Function.

The Application Interface Function provides the interface between Real Time Pseudo-Statements executed within the Application Software and the main functions of the Local Executive.

The Local Executive Proper provides the main services of the Local Executive, including control of Tasks, Events and Compool Block

The Local Executive Initialization and Recovery Function is responsible for initialization of the Local Executive after the system is loaded and for recovery or re-initialization in case of error conditions such as loss of power or other hardware-detected errors.

3.1.3.1.1 Hardware Interface Control Function

The purpose of the Hardware Interface Control Function is to isolate the hardware, i.e., the BCIU, the Timers and the processor-generated interrupts, from the rest of the Executive, so that hardware-related functions can be performed without concern for the physical details.

The primary job of this Function is to supply the Local Executive Proper with incoming Asynchronous messages in a logical form, to accept outgoing Asynchronous messages from the Local Executive Proper in a logical form, and to worry about the physical details of reception and transmission, such as buffers and BCIU registers.

The Hardware Interface Control Function processes all interrupts; therefore, it is responsible for invoking the Local Executive Initialization and Recovery Function in case of power failure, memory protection violation and other hardware-detected errors. In Master Mode, this Function must also invoke Master Executive services to respond to interrupts generated by the Timers and the Master BCIU. This Function is not responsible for supplying the BCIU with Master Instructions in Master Mode. However, this is done by the Master Executive, which runs under the control of this Function.

A general outline of the interactions of the Hardware Interface Control Function is shown in Figures 3.1.3.1.1-1 and 3.1.3.1.1-2.

3.1.3.1.2 Application Interface Function

The purpose of the Application Interface Function is to integrate Real Time Pseudo-Statements into the control structure of the Local Executive Proper. The Application Interface receives Real Time requests from the application software in its processor as well as requests from local executives in the other processors. The application Real Time requests fall into three categories: events, tasks and compools. These categories are handled by different modules of the Executive Proper. If the requests need to be transmitted to other processors, the Local Executive Proper requests the appropriate asynchronous transmission.

A general outline of the interactions of the Applications Software Interface is shown in Figure 3.1.3.1.2-1.

3.1.3.1.3 Local Executive Proper

The purpose of the Local Executive Proper is to provide the essential Local Executive services: control and processing of Tasks, Events and Compool Blocks.

The Local Executive Proper accepts Asynchronous messages from the Hardware Interface Control Function and performs the services which they request, and

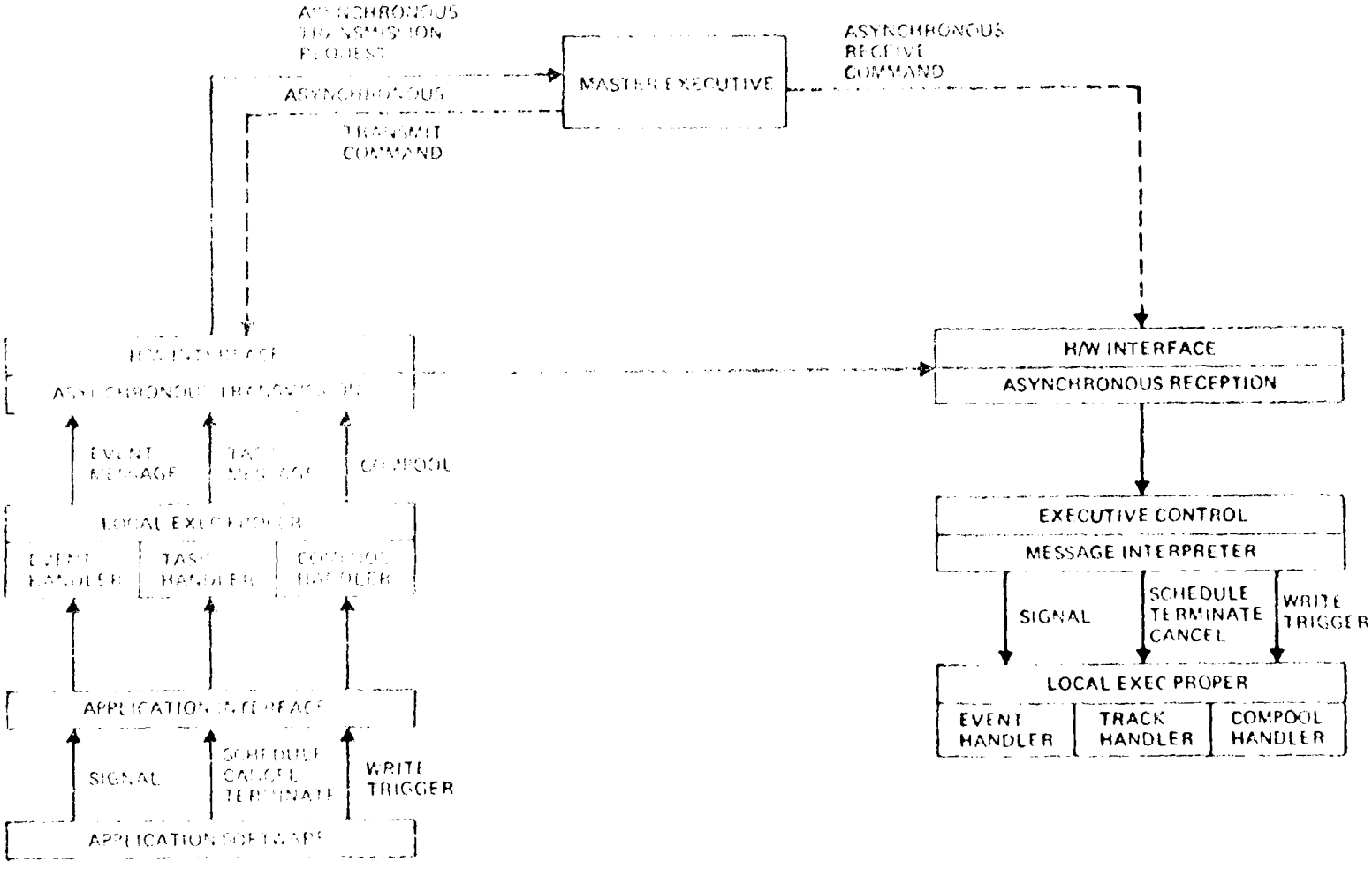


Figure 3.1.3.1.2-1. *Local Executive–Local Executive Communication*

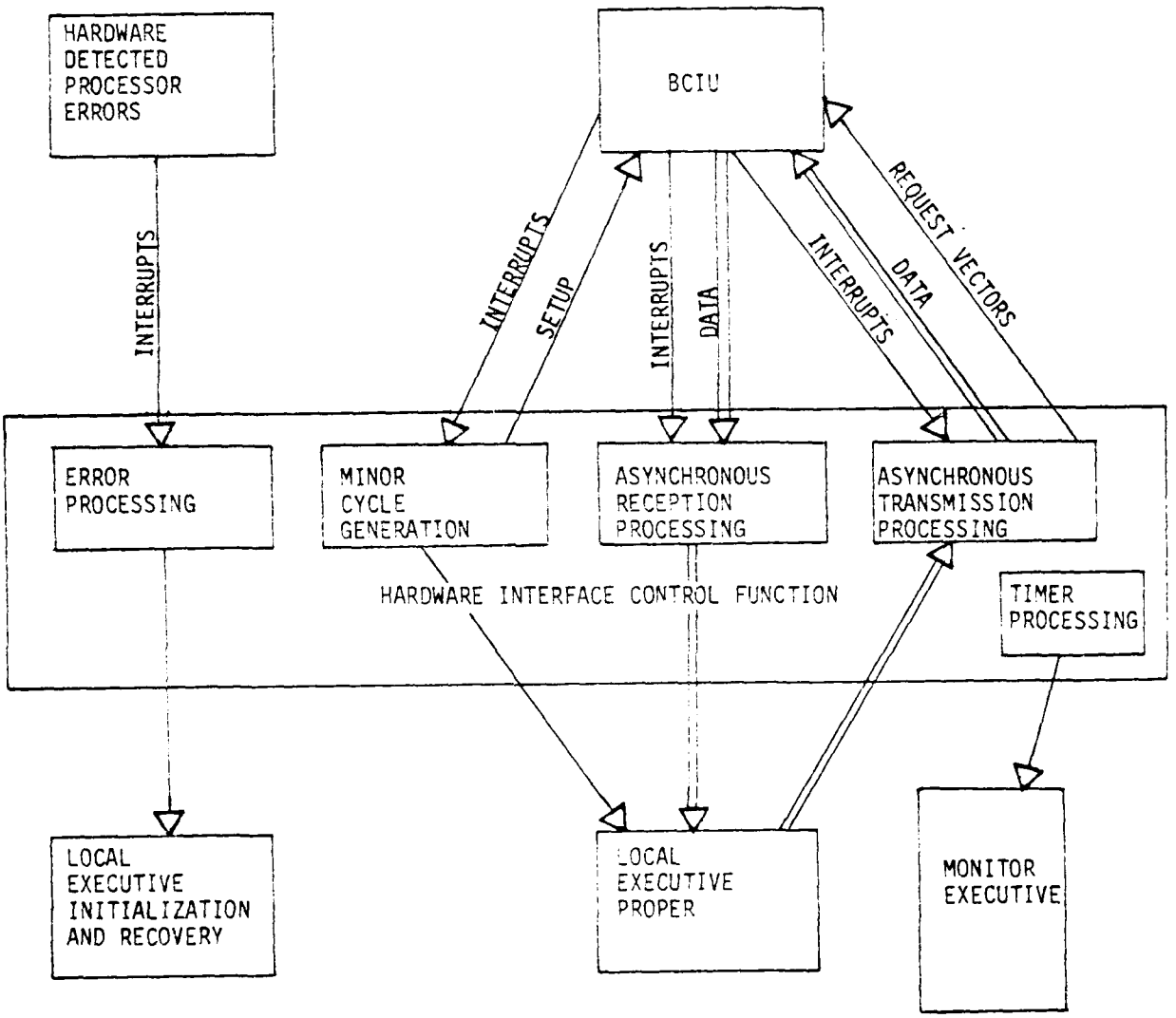


Figure 3.1.3.1.1-1 Interactions of the Hardware Interface Control Function in Remote Mode

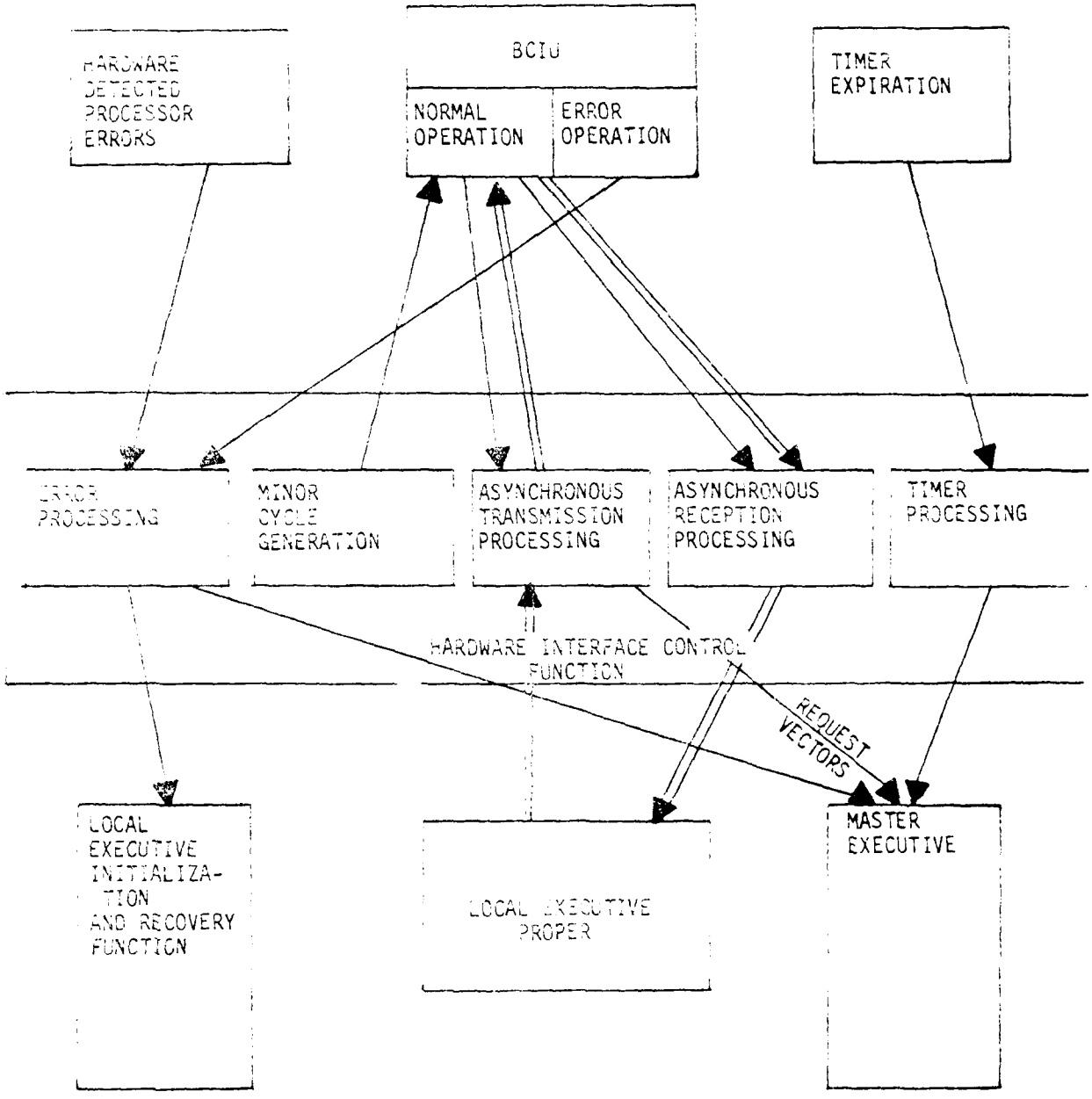


Figure 3.1.3.1.1-2 Interactions of the Hardware Interface Control Function in Master Mode

accepts and processes service requests from the Application Interface Function. It formulates and sends to the Hardware Interface Control Function Asynchronous messages requesting corresponding services in other processors or RT's.

In addition, this Function responds to new Minor Cycles by signalling Minor Cycle Events and preparing new DMA Pointers for Synchronous I/O.

A general outline of the interactions of the Local Executive Proper is shown in Figure 3.1.3.1.3-1.

3.1.3.1.4 Local Executive Initialization and Recovery Function

The Local Executive Initialization and Recovery Function provides the following services:

a. Initializes the Local Executive after it is loaded from mass memory.

b. Terminates Executive operations on detection of a "power down" condition.

c. Re-initializes the Local Executive upon power-up.

d. In case of hardware-detected processor errors, including illegal operation code, boundary alignment error, processor pointer error and processor memory protect, terminates Executive operations and sends a failure message to the Master Executive.

e. Upon detection of a Power Down condition, attempts to save the state of the processor prior to the failure.

3.1.3.2 Master Executive

The IDAMST Master Executive consists of the following major functions:

a. Master Initialization Function
b. Master Time Control Function
c. Master Synchronous Control Function
d. Master Asynchronous Control Function
e. Master Error Recovery Function
f. Mass Memory Control Function

The Master Initialization Function provides for initialization; it loads the Remote and Monitor processors from Mass Memory, and performs initial testing of the system.

The Master Time Control Function keeps track of the passage of time, and maintains proper synchronization of the various Executive services.

The Master Synchronous Control Function controls the operation of the Master BCIU in the transmission of Synchronous messages.

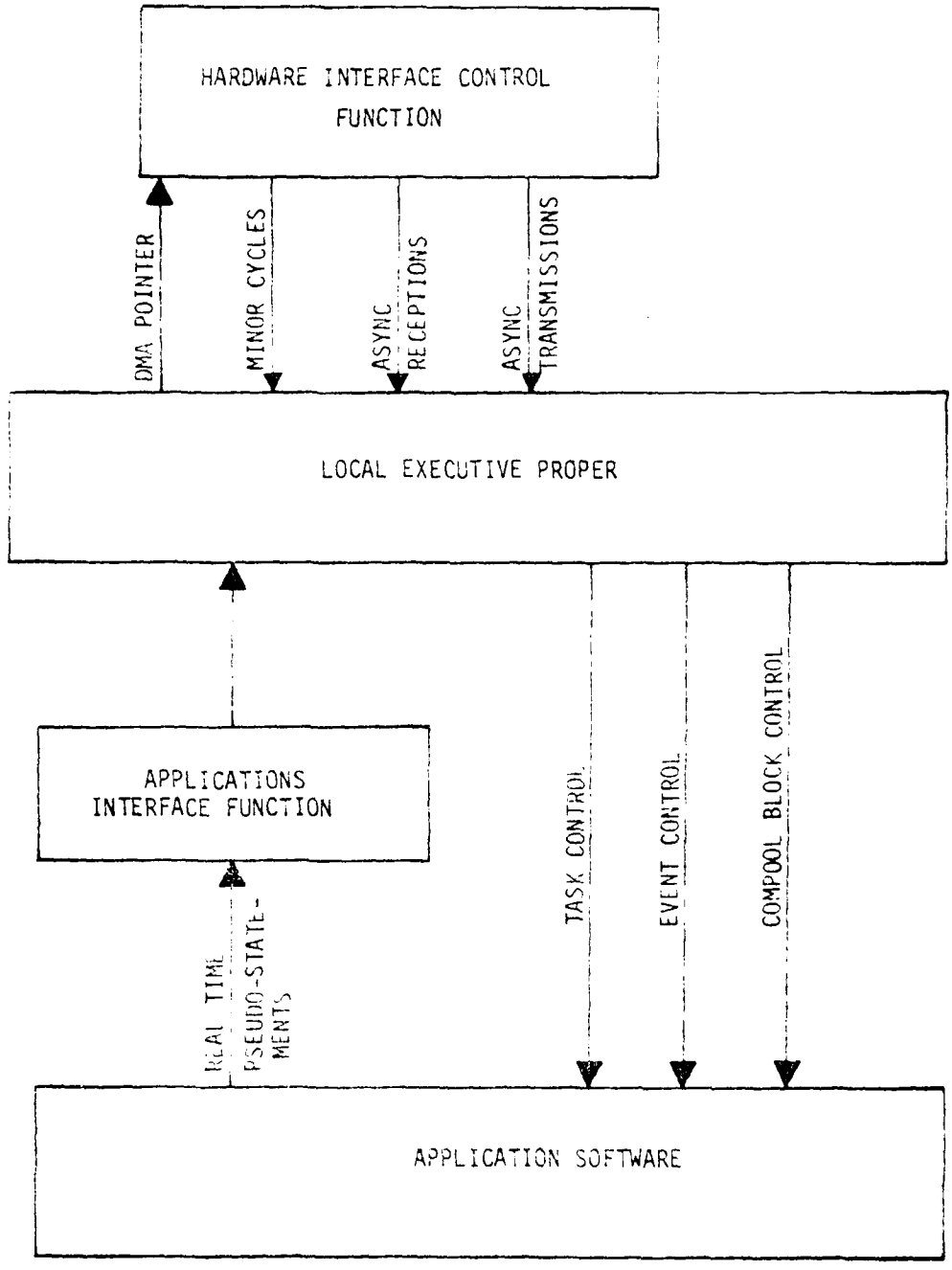


Figure 3.1.3.1.3-1 Interactions of the Local Executive Proper

The Master Asynchronous Control Function controls the operation of the Master BCIU in the transmission of Asynchronous messages.

The Master Error Recovery Function responds to routine errors encountered in the functioning of the Data Bus.

The interaction of the various functions of the Master Executive in normal operation is shown in Figure 3.1.3.2-1.

3.1.3.2.1 Master Initialization Function

The Master Initialization Function is invoked immediately after the Master Processor is loaded from Mass Memory. It is responsible for loading the Remote and Monitor processors from Mass Memory and for performing initial testing of all of the elements of the IDAMST federated system. If all systems are operative, this Function initiates normal operations.

3.1.3.2.2 Master Time Control Function

The Master Time Control Function controls the two Timers in the Master Processor. It uses Timer B to keep track of the passage of absolute time. Timer A is used to provide interrupts for synchronizing time-critical operations, i.e., setting new Minor Cycles and transmitting Critically Timed messages (see Section 3.1.2.1.8.8).

3.1.3.2.3 Master Synchronous Control Function

The Master Synchronous Control Function controls the Master BCIU in all operations which are performed repetitively rather than in response to requests by the Application Software. At the beginning of each Minor Cycle, it prepares the new Synchronous Instruction List (see 3.1.2.4.2.4) for that Minor Cycle, and causes the BCIU to execute it. When the Synchronous Instruction List is completely processed before the end of a Minor Cycle, this Function may direct the BCIU to perform other activities, such as polling for Asynchronous requests or requesting self-tests by the Remote BCIU's.

3.1.3.2.4 Master Asynchronous Control Function

The Master Asynchronous Control Function is invoked in response to Asynchronous Request Vectors received either from the BCIU or from the Local Executive in the Master processor. It directs the BCIU to perform the appropriate Asynchronous transmission.

3.1.3.2.5 Master Error Recovery Function

The Master Error Recovery Function is invoked upon detection of an error in the operation of the Multiplex System. According to the type of error, it attempts one of various re-try schemes. If all re-tries of the erroneous operation are unsuccessful, it invokes the Reconfiguration Function.

3.1.3.2.6 Mass Memory Control

The Mass Memory Control Equipment Function is used as a source of programs during system initialization, re-initialization and reconfiguration. It is also used to record Digital Integrated Test System data (DITS).

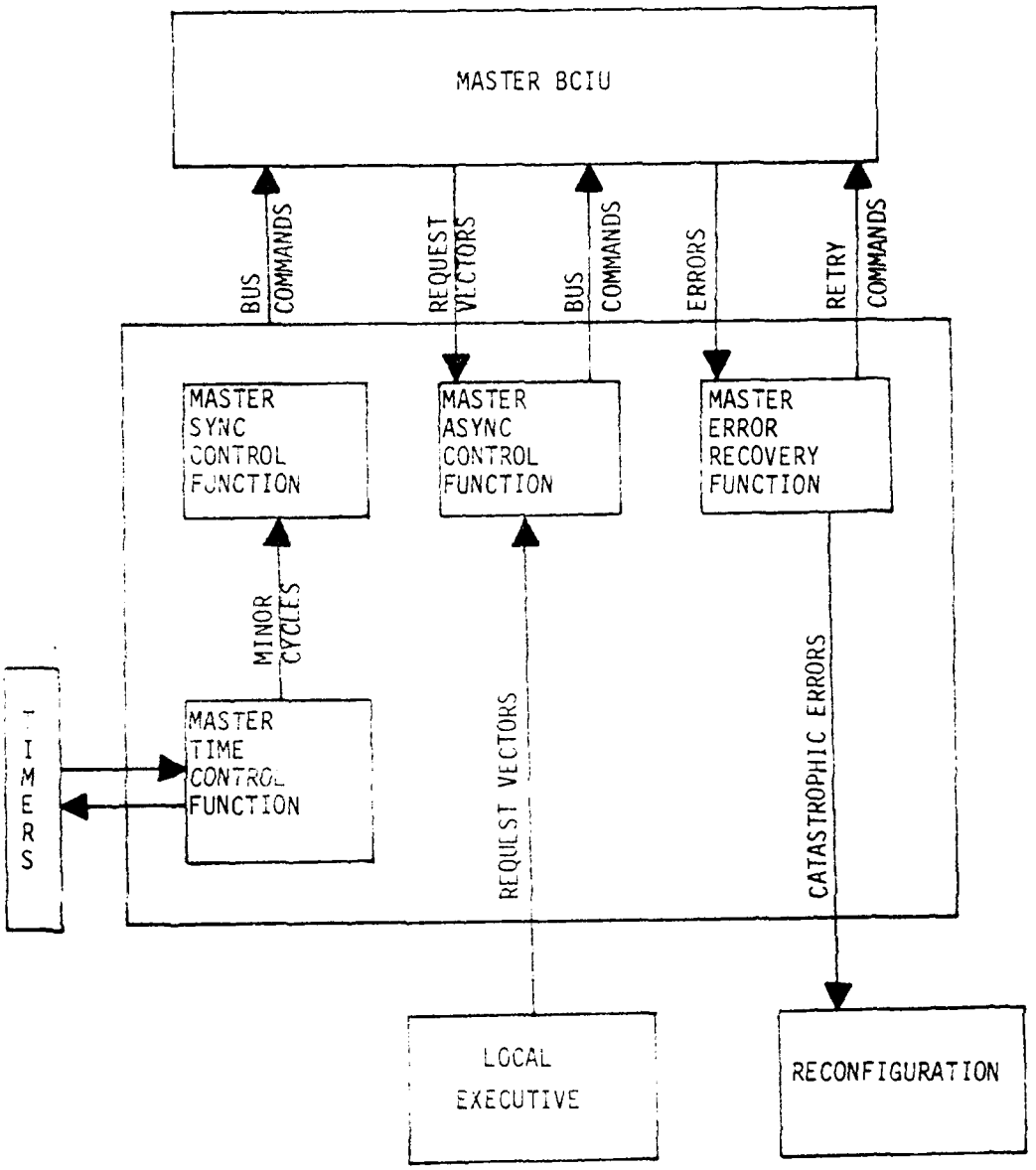


Figure 3.1.3.2-1 Interrelation of the Master Executive Functions in Normal Operation

This page left blank intentionally.

3.2 DETAILED FUNCTION REQUIREMENTS

This section specifies the detailed functional requirements of each of the functions and sub-functions of the IDAMST Executive.

3.2.1 Local Executive Functions

The Local Executive consists of four major functions:

a. The Hardware Interface Control Function
b. The Application Interface Function
c. The Local Executive Proper
d. The Local Executive Initialization and Recovery Function

The general purpose and interaction of these functions is described in Section 3.1.3.1.

3.2.1.1 Hardware Interface Control Function

The Hardware Interface Control Function is divided into sub-functions as follows:

a. The Interrupt Handling Function
b. The Asynchronous Reception Function
c. The Minor Cycle Reception Function
d. The Asynchronous Transmission Function

3.2.1.1.1 Interrupt Handling Function

The Interrupt Handling Function is always and only entered upon the reception of an Interrupt. The purpose of this routine is to:

a. Save the state of the processor prior to the interrupt.
b. Identify the cause of the interrupt.
c. Invoke the proper Executive function to service the interrupt.
d. Return to a state of normal operation.

3.2.1.1.1.1 Inputs to Interrupt Handling

The inputs to this Function are:

a. The interrupt number
b. The Internal Status Register (ISR) of the BCIU
c. The processor state prior to the interrupt, i.e., registers condition status (CS), instruction counter (IC), and Comsub Stack Pointer (see 3.1.2.1.2).
d. The identity of the last Task dispatched
e. The Privileged Mode Flag

TYPE OF INTERRUPT	FUNCTION INVOKED
Processor-Generated:	
Illegal op code Boundary alignment Processor parity	Local Executive Error Recovery
Power down	Power down
Timer A Timer B	Master Time Control (Master Mode only)
BCIU-Generated (Master and Remote):	
Async reception	Async Reception
Async transmission	Async Transmission
DMA error	Local Exec. Error Recovery
BCIU-Generated (Remote only):	
Master Function	Minor Cycle Reception
System Interrupt	Local Exec. Error Recovery
BCIU-Generated (Master only):	
Invalid Instruction	Master Reconfiguration
No Data Incomplete Data Invalid Data	Master Error Recovery
Terminal Failure	Master Reconfiguration
Status Word Error	Master Error Recovery
Status Word Exception	Master Asynchronous Control
Program Controlled	Master Synchronous Control

Table 3.2.1.1.1.2-1 Functions Invoked by the Interrupt Handling Function

3.2.1.1.1.2 Interrupt Handling Processing

The Interrupt Handling Function performs the following actions:

a. If the Privileged Mode Flag is on, it saves the prior state of the processor in a local Save Area; otherwise, it saves the prior state in the Task Table B entry of the Task which was interrupted.
b. It identifies the cause of the interrupt, reading the ISR of the BCIU if necessary.
c. It invokes the appropriate Executive function to service the interrupt.
d. It enables interrupts.
e. If the Privileged Mode Flag is on, it returns to the state prior to the interrupt; otherwise, it sets the Privileged Mode Flag and Executive Control Function.

The Executive function invoked for each type of interrupt is shown in Table 3.2.1.1.1.2-1.

3.2.1.1.1.3 Outputs from Interrupt Handling

The outputs from this Function are:

a. The identity of the interrupt
b. The Save Area in the Task Table B entry of the interrupted Task (see 3.1.2.4.1.3).

3.2.1.1.2 Asynchronous Reception Function

The purpose of the Asynchronous Reception Function is to accept an incoming Asynchronous message from the BCIU, to enqueue the message for processing by the Local Executive Proper, and to prepare for a new reception.

In addition, this Function intercepts messages requesting re-transmission of the last transmitted message, and invokes the Asynchronous Transmission Function to service them.

This Function is always and only invoked by the Interrupt Handling Function upon the occurrence of an interrupt indicating completion of an Asynchronous reception.

3.2.1.1.2.1 Inputs to Asynchronous Reception

The sole input to this Function is the Reception Queue. This Queue consists of:

a. A fixed number of 33 word buffers for receiving Asynchronous messages.
b. A pointer to the first buffer in the Queue.
c. A pointer to the last buffer in the Queue.
d. The Request Pending Flag.

The Reception Queue is a First in First Out queue. Buffers are filled by the BCIU, and removed from the Queue by the Local Executive Control Function. The "first buffer" pointer points to the next item on its way out of the Queue, i.e., the buffer succeeding the last one fully processed by the Local Executive Proper.

Normally, the "last buffer" pointer points to the last buffer filled by an Asynchronous reception from the BCIU. However, when the Asynchronous Reception Function is invoked, the buffer succeeding the one pointed to by the "last buffer" pointer has just been filled with an Asynchronous message.

If any of the buffers contains an Asynchronous message which has not been processed by the Executive Proper, the Request Pending Flag is on; otherwise it is off.

The buffers are considered to be arranged cyclically. That is, the buffer which is physically first is considered to succeed the buffer which is physically last. A typical configuration of the Queue, a four-buffer system with two messages in the Queue, is shown in Figure 3.2.1.1.2.1-1.

3.2.1.1.2.2 Asynchronous Reception Processing

The processing of this Function is shown in Figure 3.2.1.1.2.2-1.

Note that if all buffers are full, the Asynchronous Reception Function does not reinstate bus operations.

3.2.1.1.2.3 Outputs from Asynchronous Reception

The outputs from this Function are:

a. The updated Reception Queue (see 3.2.1.1.2.1).
b. The Continue Bit in the BCIU
c. Asynchronous Reception Pointers in the DMA Pointer Blocks (See 3.1.2.4.1.1).

3.2.1.1.3 Minor Cycle Reception Function

The purpose of the Minor Cycle Reception Function is to accept and to enqueue new Minor Cycle numbers received from the BCIU. This Function is always and only invoked by the Interrupt Handling Function upon an interrupt indicated by the reception of a Minor Cycle Mode Command. In Master Mode, this Function does not operate, Minor Cycles being directly enqueued by the Master Executive.

3.2.1.1.3.1 Inputs to Minor Cycle Reception

The sole input to this Function is the Minor Cycle Register in the BCIU.

3.2.1.1.3.2 Minor Cycle Reception Processing

The Minor Cycle Reception Function performs the following operations:

a. It reads the new Minor Cycle number from the BCIU.
b. It sets the Minor Cycle Pending Flag on.

Note that this Function does not reinstate bus operations.

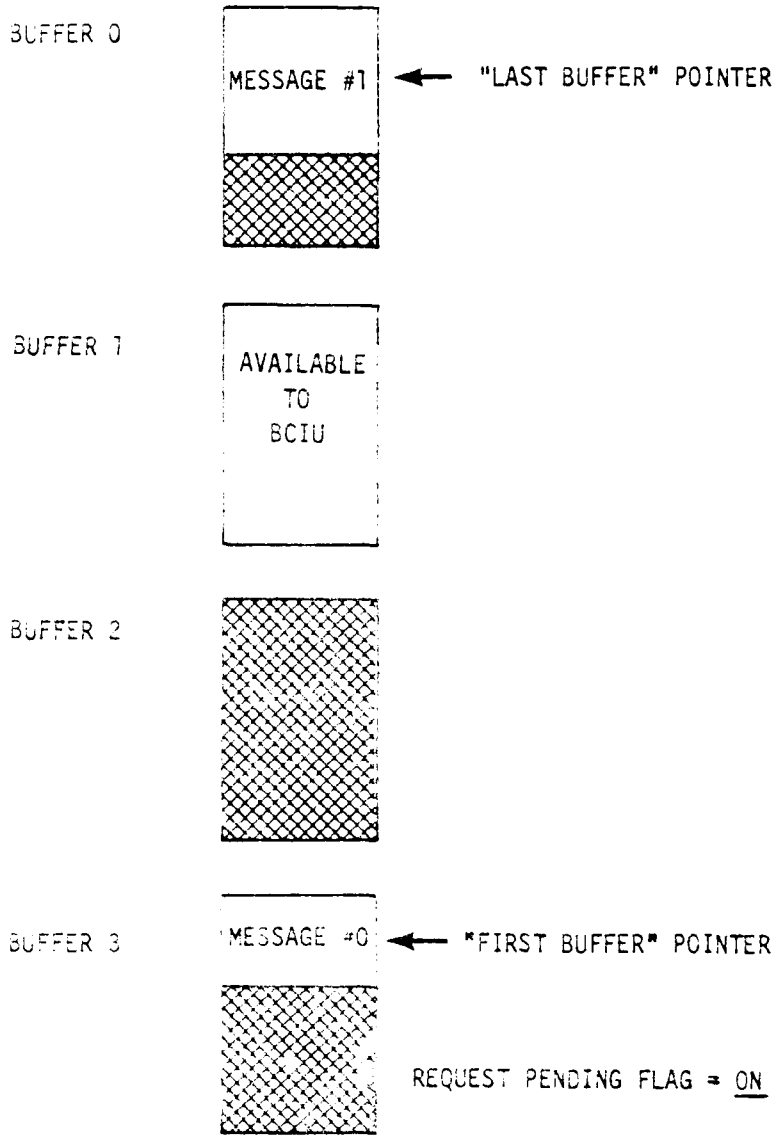


Figure 3.2.1.1.2.1-1 Example of Reception Queue

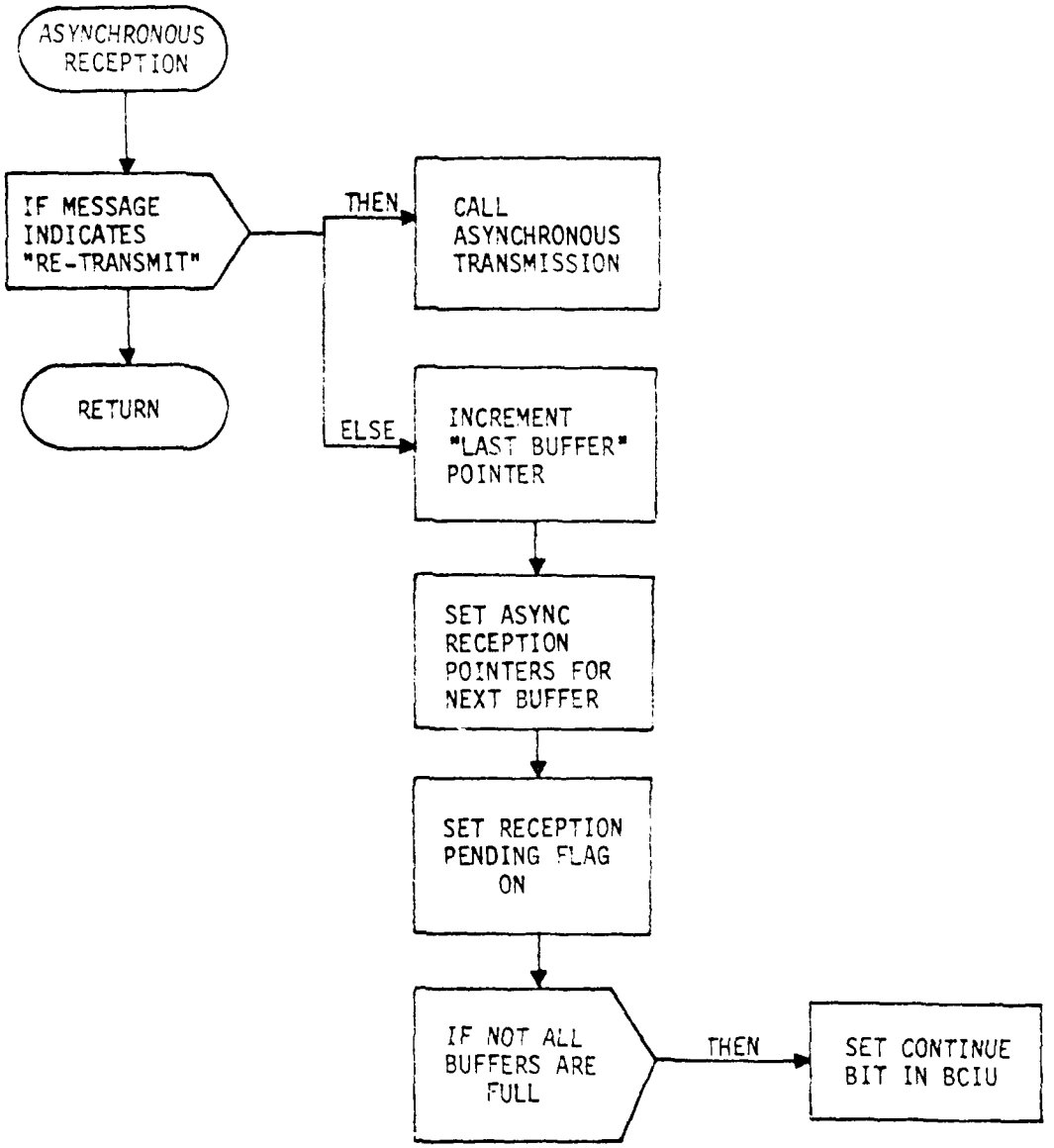


Figure 3.2.1.1.2.2-1 Processing of Asynchronous Reception Function

3.2.1.1.3.3 Outputs from Minor Cycle Reception

The outputs from this Function are:

a. The new Minor Cycle number
b. The Minor Cycle Pending Flag

3.2.1.1.4 Asynchronous Transmission Function

The purpose of the Asynchronous Transmission Function is to accept outgoing Asynchronous messages enqueued by the Local Executive Proper and to prepare them for transmission by the BCIU.

This Function can be invoked from three places:

a. By the Interrupt Handling Function upon completion by the BCIU of an Asynchronous transmission;
b. By the Asynchronous Reception Function upon reception of a message requesting re-transmission of the last message transmitted; or
c. By the Local Executive Proper to request initiation of an asynchronous BCIU transmission.

3.2.1.1.4.1 Inputs to Asynchronous Transmission

Inputs to this Function are:

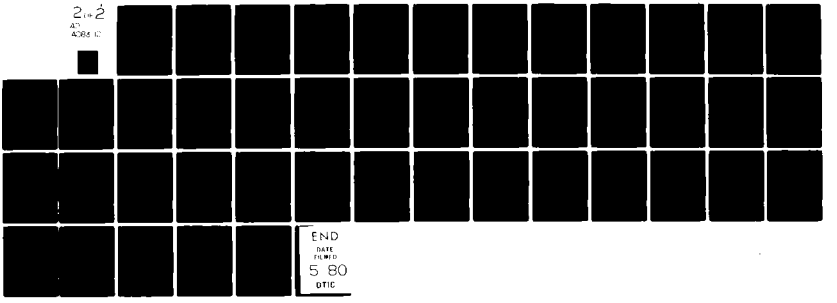
a. Indication of who invoked it.
b. The Transmission Queue

The Transmission Queue is a First In First Out queue, containing all Asynchronous transmissions which have been enqueued by the Local Executive Proper, but not yet discarded by the Asynchronous Transmission Function. The Asynchronous Transmission Function does not discard a transmission until the next transmission has been completed by the BCIU.

The Transmission Queue consists of:

a. A fixed number of message descriptor Blocks.
b. A Last Transmitted Pointer (LTP), which usually points to the Descriptor Block of the last message transmitted by the BCIU.
c. A Current Transmission Pointer (CTP), which usually points to the Descriptor Block of the message currently set up for transmission by the BCIU.
d. A Final Descriptor Pointer (FDP), which points to the Descriptor Block of the message most recently enqueued by the Local Executive Proper.
e. A Fixed Transmission Buffer Area.
f. A First Free Buffer Pointer (FFBP), which points to the first free word in the Transmission Buffer Area.
g. A First Used Buffer Pointer (FUBP), which points to the first word in the Transmission Buffer Area used for enqueuing of messages by the Local Executive Proper.

AD-A083 112 BOEING AEROSPACE CO SEATTLE WA BOEING MILITARY AIRPL--ETC F/G 9/2
COMPUTER PROGRAM DEVELOPMENT SPECIFICATION FOR IDAMST OPERATION--ETC(U)
NOV 76 F33615-76-C-1099
UNCLASSIFIED SPEC-SB-4041 AFAL-TR-76-208-ADD-1 NL



When the Asynchronous Transmission Function is invoked upon completion of an Asynchronous transmission by the BCIU, the CTP points to the Descriptor Block of the message just transmitted.

When the Transmission Queue is empty, that is, there are no messages waiting for transmission, the LTP and the FTP both point to the Descriptor Block of the last message transmitted, and the CTP points to the next Descriptor Block, even though there is no message set up for BCIU transmission.

When the BCIU is set up for re-transmission of a message previously transmitted, CTP=LTP.

Each Message Descriptor Block consists of:

a. A Request Vector
b. An Asynchronous ID word
c. A pointer into the Transmission Buffer Area.

The Request Vector is the Request Code for transmission of the message.

The Asynchronous ID word is the word to be appended to the beginning of the message to identify it to the receiving Local Executive. If this message is to be sent to an RT, this word is octal 177777, and will not be appended to the message.

The buffer pointer points to a buffer allocated within the Transmission Buffer Area holding the data to be transmitted. The first two words of each buffer are blank, to allow for the Minor Cycle Tag and the Async ID; the data begins on the third word. If a message has no associated data, i.e., is an interprocessor Service Request, the buffer pointer is zero and no buffer is allocated for the message. It is possible for a series of messages to point to the same buffer, as when a single compool block is sent to more than one processor.

The FUBP points to the first word of the Buffer Area that may <u>not</u> be used for enqueuing new messages, i.e., that contain data that has either not yet been transmitted by the BCIU or may have to be retransmitted by the BCIU. The FFBP points to the first word of the Buffer Area that <u>may</u> be used for enqueuing messages by the Local Executive Proper. FFBP+33 may never be outside of the Buffer Area, and FFBP+33 $\leq$ FUBP at all times, since any message may be up to 33 words, including the Tag Word.

A typical configuration of the Transmission Queue is shown in Figure 3.2.1.1.4.1-1. This system has eight Message Descriptor Blocks, six of which describe messages pending transmission. Note that message #3 and message #4 share the same data, while message #2 has no data, and hence, no buffer.

3.2.1.1.4.2 Asynchronous Transmission Processing

The Asynchronous Transmission Function performs the following actions:

a. If invoked to perform a re-transmission, it sets CTP to previous message in Queue.

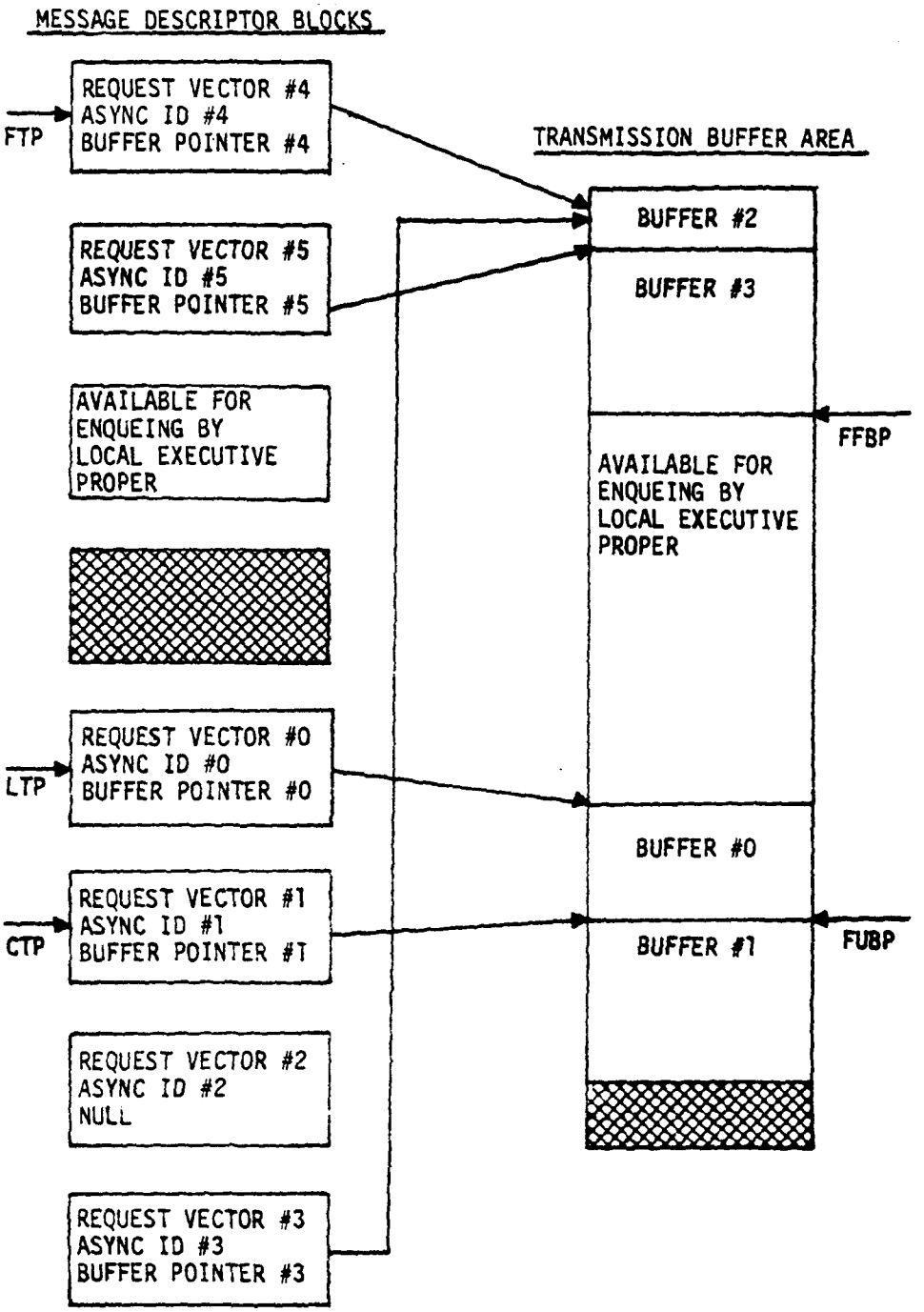


Figure 3.2.1.1.4.1-1 Example of Transmission Queue

b. If invoked upon completion of Asynchronous transmission, it sets CTP to the next message in the Queue, discards the previous message in the Queue and sets the FUBP forward if a buffer has been discarded.
c. If there is a message in the Queue waiting for transmission, it copies the Asynchronous Transmission Pointers in the DMA Pointer Blocks to that buffer, and sends the appropriate Request Vector either to the BCIU or, in Master Mode, the the Master Asynchronous Control Function. Messages with a null Buffer Pointer in their Descriptor Blocks are transmitted from a local buffer.
d. If the BCIU is "busy", the Continue Bit in the BCIU is set.

The output of the processing is shown in Figure 3.2.1.1.4.2-1.

3.2.1.1.4.3 Outline of Asynchronous Transmissions

The outputs from this Function are:

a. The updated Transmission Queue (see 3.2.1.1.4.1).
b. The Asynchronous Transmission Pointers in the DMA Pointer Blocks (see 3.1.2.4.1.1).
c. The Continue Bit in the BCIU.
d. A Request Vector (to the Status Code Register in Remote Mode; to the Master Asynchronous Control Function in Master Mode).

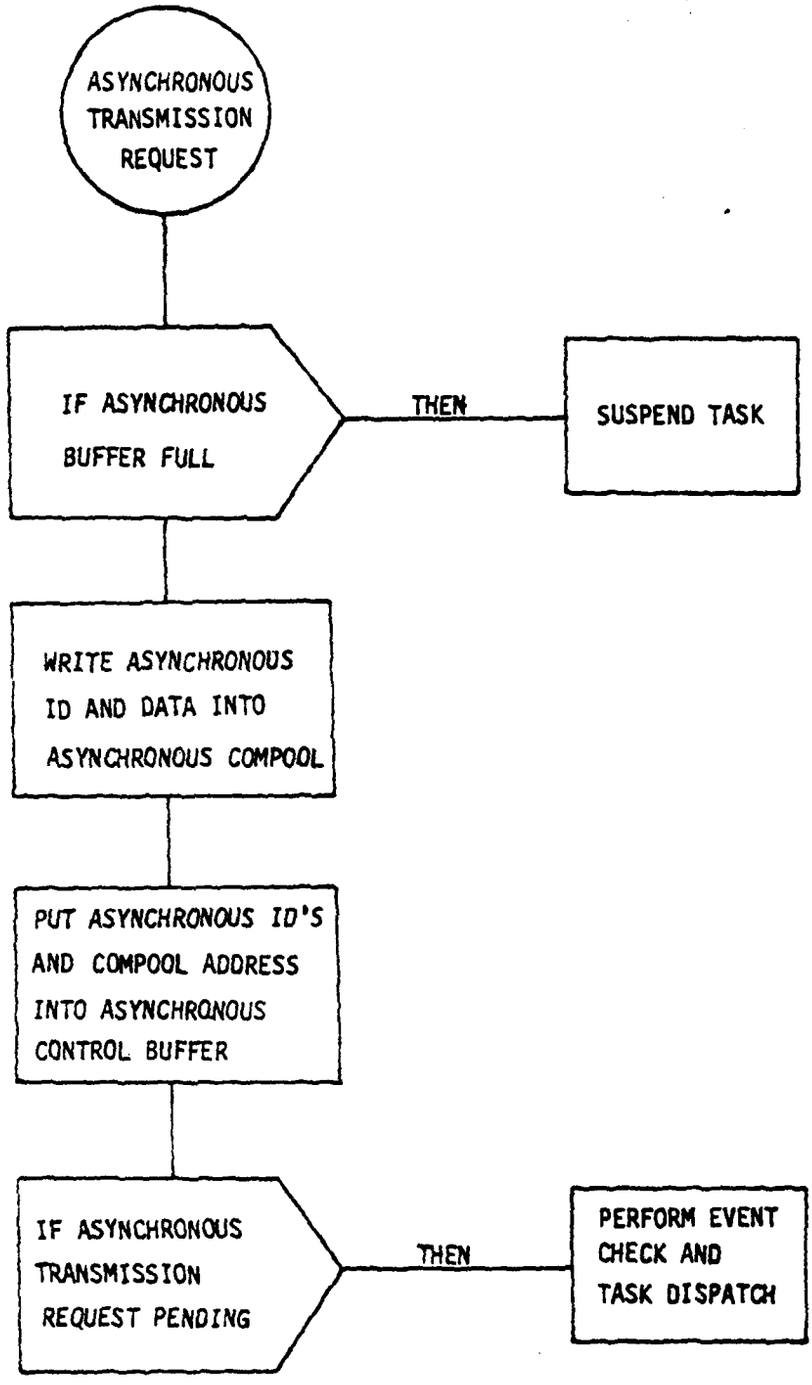


Figure 3.2.1.1.4.2-1 Asynchronous Transmission

3.2.1.2 Application Interface Function

The Application Interface Function serves to save the processing state of the invoking task, to change to the privileged state of the IDAMST processor, and to invoke the specific Local Executive Function that was requested by the Real-Time Pseudo-Statement from the Applications Task. The Application Interface Function also does not save the Task state in the case of three specific Executive Service Requests (ESR's). For those three Intrinsic functions, the request is satisfied and control returned to the requesting Task.

The three specific ESR's contained in the Application Interface Function are EREAD, INVOKED, and TIME.

3.2.1.3.1 Executive Service Routines

There are fifteen Executive Service Routines, one for each type of Real Time Pseudo-Statements, including the four separate types of Wait Statements (see 3.1.2.1.8).

3.3.1.2.1.1 Inputs to Executive Service Routines.

The inputs to the various Executive Service Routines are shown in Table 3.2.1.2.1.1-1.

3.2.1.2.1.2 Executive Service Routine Processing

Each Executive Service Routine performs the following actions:

a. It sets the Privileged Mode Flag
b. It saves the state of the invoking Task in the Task Table B.
c. It calls the appropriate Function in the Local Executive Proper.

The Function invoked by each Executive Service Routine, and the parameters passed to it, are shown in Table 3.2.1.2.1.2-1. The interfaces and some of the support-ind data are shown in Table 3.2.1.2.1.2-2.

3.2.1.2.1.3 Outputs from Executive Service Routines

The output from each Executive Service Routine is the updated status of the Application Software requested by the corresponding Pseudo-Statement. See 3.1.2.1.8 for details.

3.2.1.2.2 Application Interface - Intrinsic Functions

Application Interface contained Functions are short functions which can index into the data structures used by the Local Executive Proper routines discussed in 3.2.1.3.

ROUTINE	INPUTS
Schedule	Task Table A entry of Task to be Scheduled
Cancel	Task Table A entry of Task to be Cancelled
Terminate	Task Table A entry of Task to be Terminated
Wait:	
Absolute Time	Absolute Time
Relative Time	Relative Time
Latched	Event Table entry of Event to be waited on; desired value
Unlatched	Event Table entry of Event to be waited on; desired value
Signal	Event Table entry of Event to be signalled; desired value
Read	DDB for Compool Block to be Read; Local Copy to be read into
Write	DDB for Compool Block to be Written; Local Copy to be written from
Trigger	DDB for Compool Block to be Triggered; Local Copy to be written from; time to go
ERead	Event number
Invoke	Task ID
IO Device	Number of device to be manipulated; off or on state desired, reconfiguration flag.
Time	Null

Table 3.2.1.2.1.1-1 Inputs to Executive Service Routines

ROUTINE	FUNCTION INVOKED	PARAMETERS
Schedule	Schedule	1) Task Table A entry
Cancel	Cancel/Terminate	1) Task Table A entry 2) Cancel/Terminate Flag = Cancel
Terminate	Cancel/Terminate	1) Task Table A entry 2) Cancel/Terminate Flag = Terminate
Wait: Absolute Time Relative Time	Wait	1) Time/Event Flag = Time 2) Absolute time to go
Latched Unlatched	Wait	1) Time/Event Flag = Event 2) Event Table entry 3) Desired value of Event 4) Latched/Unlatched Flag
Signal	Event Handling	) Internal/External Flag = Internal 2) Event Table entry 3) Desired value of Event
Read Write	Compool Block Handling	1) Internal/External Flag = Internal 2) Trigger Flag = off 3) DDB 4) Local Copy
Trigger	Compool Block Handling	1) Internal/External Flag = Internal 2) Trigger Flag = on 3) DDB 4) Local Copy 5) Time to go
IO Device	IO Device	1) Device number 2) Flag = ON/OFF 3) Reconfiguration Flag
ERead	Application Interface	1) ERead/Time/Invoked Flag = ERead 2) Value of Event
Invoked	Application Interface	1) ERead/Time/Invoked Flag = Invoked 2) Task Table A entry 3) Value of Invoked
Time	Application Interface	1) ERead/Time/Invoked Flag = Time 2) Time value

Table 3.2.1.2.1.2-1 Functions Invoked by Executive Service Routines

3.2.1.2.2.1 ERead

ERead interrogates and returns the value of a specific event. This function properly belongs with the event handler function, but is separated from the event handler because of efficiency requirements. The inputs are described in Table 3.2.1.2.1.2-1.

3.2.1.2.2.2 Invoked

Invoked interrogates and returns the status of a specified task. This function properly belongs as part of a task handling function such as Checker (3.2.1.3.4), which determines the status of a task, but is separated because of efficiency considerations. The inputs are described in Table 3.2.1.2.1.2-1.

3.2.1.2.2.3 Time

Time is a function that returns the elapsed time since system initialization. There are no input parameters; the processing is a readout of the processor clock register and appending that value to the cumulative value since system initialization. The output of Time is the cumulative time since system initialization.

3.2.1.2.3 Executive Service Return Function

The Executive Service Return Function is called by all Executive Service Routines immediately before they relinquish control. The purpose of this Function is to determine whether there is a need to perform additional Local Executive Functions, either as a result of Asynchronous messages or Minor Cycles received while in Privileged Mode or because the Service itself requested further Services. If there are further services to perform, this Function generates a pseudo-interrupt; otherwise, it resets the Privileged Mode Flag and returns to its caller.

3.2.1.2.3.1 Inputs to Executive Service Return

The inputs to this Function are:

a. The Reception Pending Bit (see 3.2.1.1.2.1)
b. The Minor Cycle Pending Bit (see 3.2.1.1.3.1)
c. The Event Queue (see 3.2.1.3.1.1)
d. The identity of the last Task dispatched

3.2.1.2.3.2 Executive Service Return Processing

The processing of the Executive Service Return Function is shown in Figure 3.2.1.2.3.2-1.

3.2.1.2.3.3 Outputs from Executive Service Return

The sole output of this Function is the Save Area of the last Task dispatched (see 3.1.2.4.1.14).

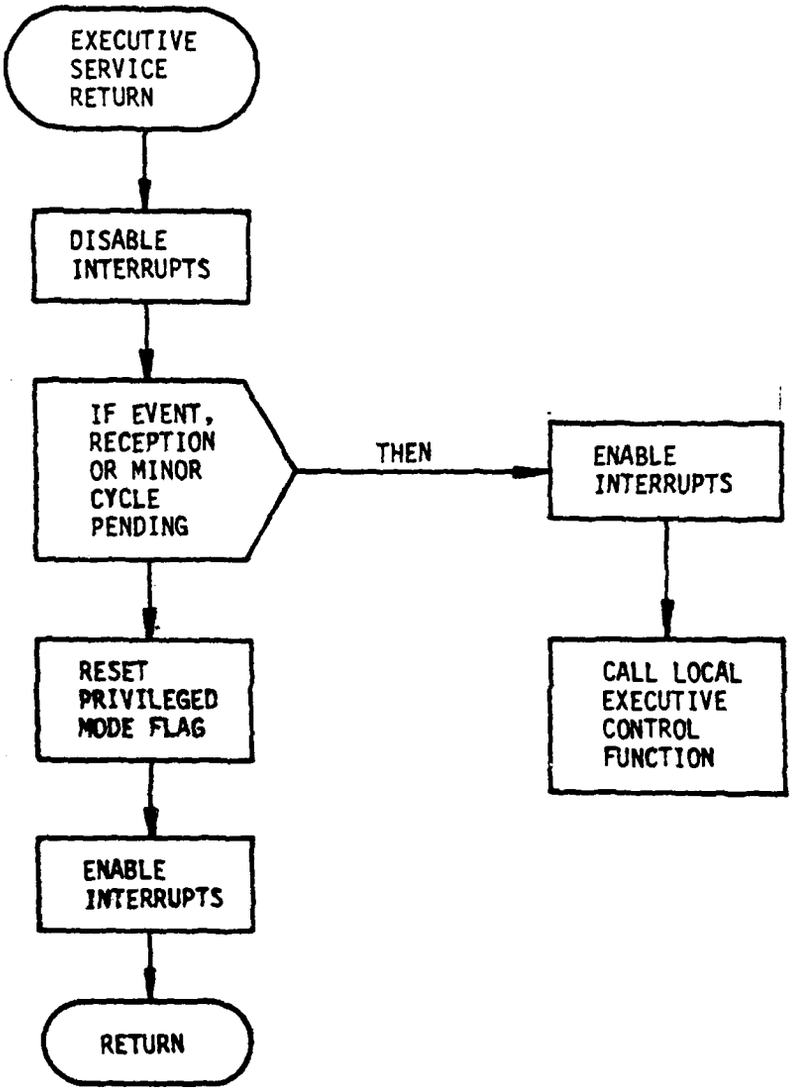


FIGURE 3.2.1.2.3.2-1. EXECUTIVE SERVICE RETURN PROCESSING

3.2.1.3 Local Executive Proper

The Local Executive Proper consists of ten subfunctions:

a. Local Executive Control Function
b. Minor Cycle Setup Function
c. Event Handling Function
d. Task Checking Function
e. Task Scheduling Function
f. Task Termination/Cancellation Function
g. Wait Function
h. Compool Block Handling Function
i. Dispatch Function
j. IO Device Function

3.2.1.3.1 Local Executive Control Function

The Local Executive Control Function maintans the proper sequencing of the subfunctions of the Local Executive Proper. This Function is called either by the Application Interface Function after a Real Time Pseudo-Statement has been serviced, or by the Hardware Interface Function after an interrupt has been serviced. The Local Executive will return control directly to the calling or interrupted task if none of the following is true:

a. There is a Minor Cycle pending.
b. There is an Asynchronous Message Pending.
c. The Event Queue is non-zero.
d. The highest priority dispatchable task is higher priority than the last dispatched task. Privileged tasks are considered to be highest priority tasks.

When a Normal task makes an Executive Service Request or is interrupted, and any of the conditions shown above are true, the Local Executive will service the condition and then call the Dispatcher.

The Event Queue is used by the subfunctions of the Local Executive Proper when they wish to set an Event to a certain value. It is necessary to enqueue the Event and its desired value rather than directly calling the Event Handling function to avoid recursion. The Event Queue is serviced on a Last In First Out basis. Each item in the Queue contains:

a. a pointer to the Event Table entry of an Event, and
b. the desired value of the Event.

3.2.1.3.1.2 Local Executive Control Processing

A general outline of the processing of the Local Executive Control Function is shown in Figure 3.2.1.3.1.2-1.

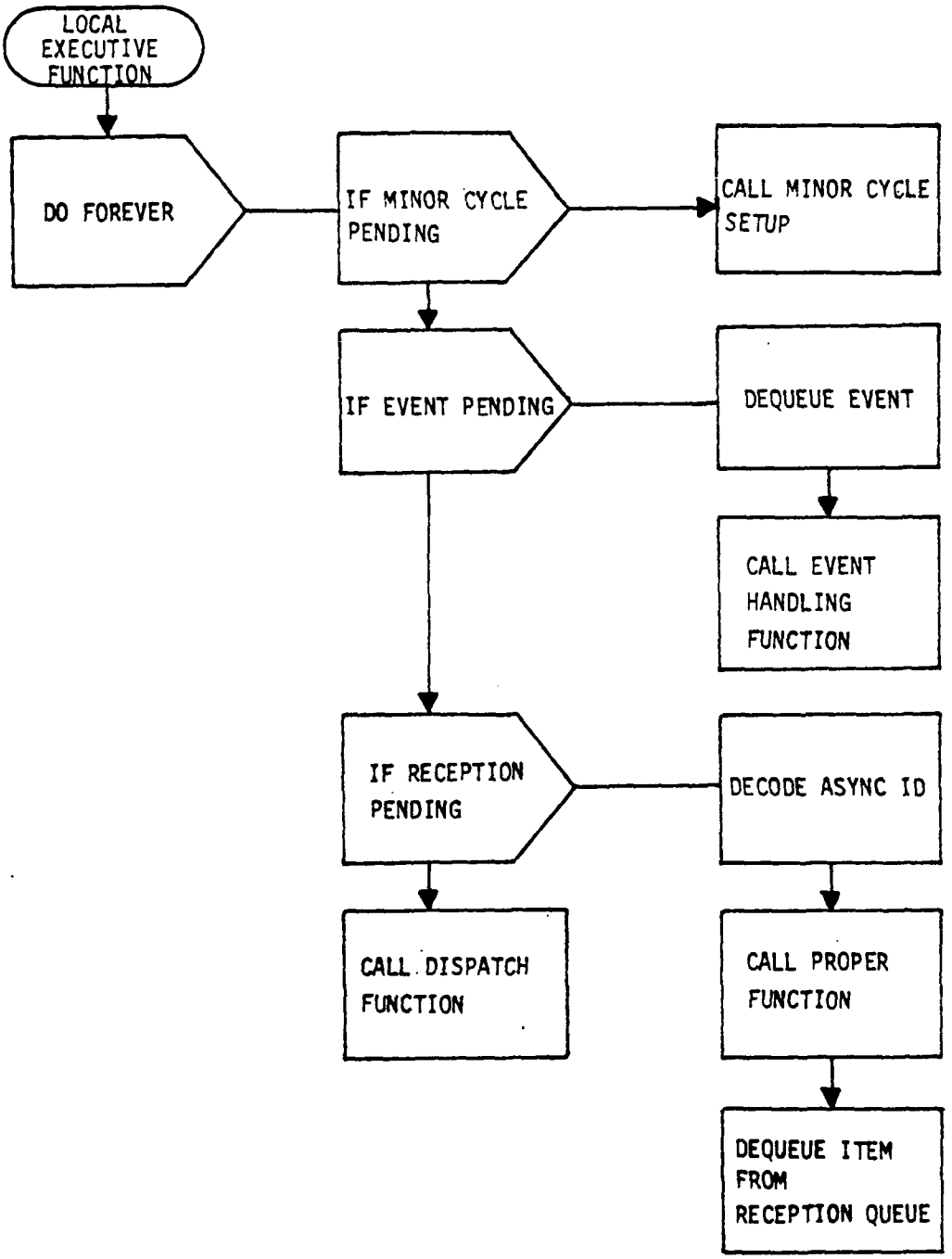


Figure 3.2.1.3.1.2-1 Local Executive Control Processing

Note that an Event must be dequeued before it is processed, because the Event Queue is Last In First Out, whereas an Asynchronous reception must be dequeued after it is processed because the information in the buffer in the Reception Queue may be used in the course of processing the reception.

After the Local Executive Control Function dequeues an Event from the Event Queue, it passes the following parameters to the Event Handling Function:

a. Internal/External Flag = Internal
b. Pointer to Event Table entry for Event
c. Desired value of Event

The Asynchronous ID which this Function uses to determine the type of each Asynchronous message in the Reception Queue may be either a Transmit Word, indicating that the message was sent by an RT, or a "true" Asynchronous ID produced by the Local Executive which sent the message. If it is a Transmit Word, the Local Executive Control Function uses the TOAD and the SNAKE (see 3.1.2.4.1) to determine the DDB associated with the message, and calls the Compool Block Handling Function. Otherwise, it determines the type of message from the OP Code field of the Async ID, and determines the parameters to pass to the specified function by the Parameter field of the ID. The types of Async ID's, the Functions invoked to service them, and the parameters passed to those Functions are shown in Table 3.2.1.2.1.2-1.

3.2.1.3.1.3 Outputs from Local Executive Control

This Function has no outputs, since it is never exited.

3.2.1.3.2 Minor Cycle Setup Function

The Minor Cycle Setup Function performs the processing necessary to set up the environment of a new Minor Cycle. This Function is always and only invoked by the Local Executive Control Function upon detection of a pending Minor Cycle.

3.2.1.3.2.1 Inputs to Minor Cycle Setup

The inputs to this Function are:

a. The new Minor Cycle Number
b. The SYNPTR and SYNPTR Index Table (see 3.1.2.4.2)
c. The Minor Cycle Event Generation Table (see 3.1.2.4.1.5)
d. The Chain of Tasks waiting on time (see 3.2.1.3.7.2)

3.2.1.3.2.2 Minor Cycle Setup Processing

The Minor Cycle Setup Function performs the following actions:

a. If new Minor Cycle Number is out of sequence, call the Error Recovery Function.
b. Set Minor Cycle Number to new value.
c. Set Base Address Register to appropriate DMA Pointer Block.
d. Set Continue Bit in BCIU.
e. Set up DMA Pointer Block for next Minor Cycle using the SYNPTR and SYNPTR Index Table.
f. Enqueue the appropriate Minor Cycle Events, using the Minor Cycle Event Generation Table.
g. Look at the first Task waiting on time. If it is to go on this Minor Cycle, take it out of the Chain and make it Dispatchable.
h. Repeat Step g until there are no Tasks waiting on this Minor Cycle.

The outline of these functions is given in Figure 3.2.1.3.2.2-1.

3.2.1.3.2.3 Outputs from Minor Cycle Setup

The outputs from this Function are:

a. The Minor Cycle Number
b. The new DMA Pointer Blocks (see 3.1.2.4.1.1)
c. The Base Address Register in the BCIU
d. The Continue Bit in the BCIU
e. The Event Queue with the appropriate Minor Cycle Events enqueued.
f. The updated Task Table B entries of all Tasks which were waiting for this Minor Cycle.

3.2.1.3.3 Event Handling Function

The event handling function will be called for regular event signalling, block update, or task completion events declared by tasks in the processor. This Function may be invoked "internally" either by the Application Interface Function or by the Local Executive Control Function, to service an Event Signal request emanating from within the processor. Or it may be invoked "externally" by the Local Executive Control Function to service an Event Signal request from another processor.

3.2.1.3.3.1 Inputs to Event Handling

The inputs to this Function are:

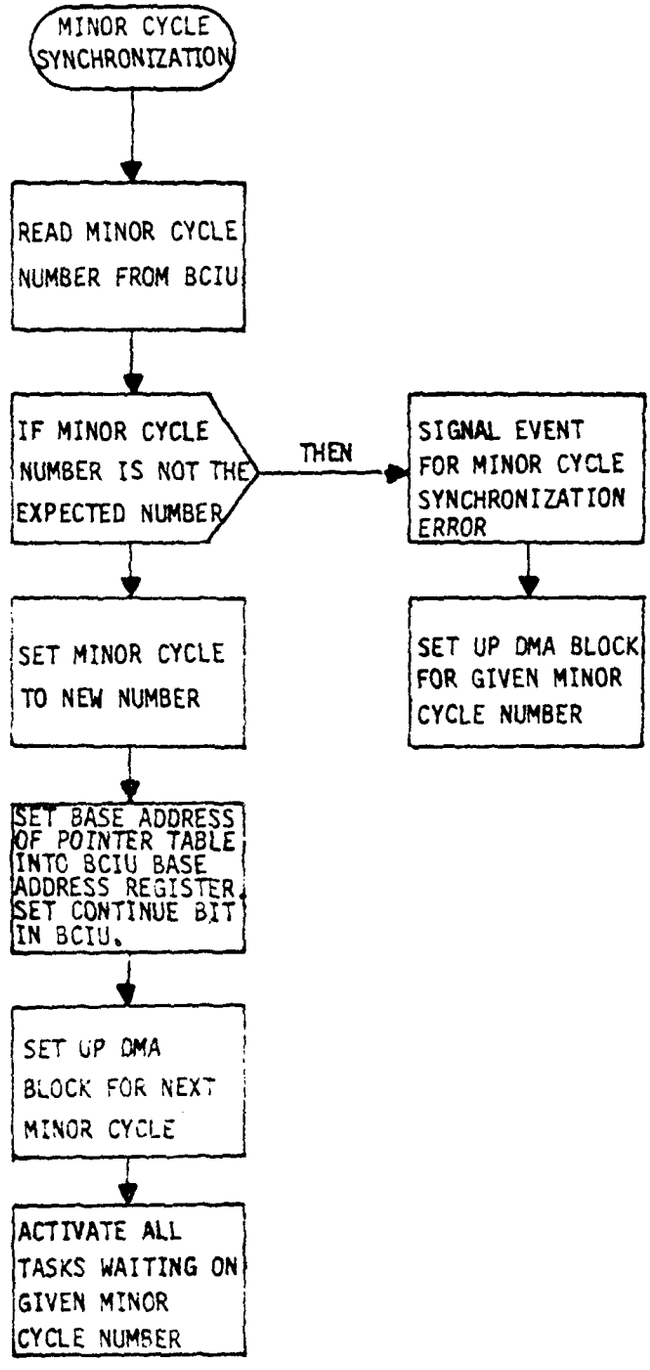


Figure 3.2.1.3.2.2-1 Minor Cycle Synchronization

a. The Internal/External Flag
b. The Event Table entry of an Event
c. The desired value of that Event

3.2.1.3.3.2 Event Handling Processing

The Event Handling Function sets the value of an Event in the Event Table, formulates and enqueues Asynchronous messages to other processors to set their copies of that Event, sets the Conditions in the Task Table B entries of Tasks with the Event in their Condition Set, and invokes the Task Check Function to check the status of those Tasks. The processing of the Event Handling Function is shown in Figure 3.2.1.3.3.2-1.

3.2.1.3.3.3 Outputs from Event Handling

The outputs from this Function are:

a. The Transmission Queue, with requests to other processors to set their copies of this Event.
b. The updated Event Table entry of the Event.
c. The updated Task Table B entry of each Task with this Event in its condition set.
d. The updated Task Table B entry of all Tasks waiting on the current value of the Event.

3.2.1.3.4 Task Checking Function

The purpose of the Task Checking Function is to check whether a Task should be changed from Inactive to Active State, and if so, to change its State and perform all associated actions.

This Function is invoked in the following circumstances:

a. When a Task is Scheduled.
b. When an Event in the Task's Condition Set is Signalled.
c. When a Task either ends or is forcibly Terminated.

3.2.1.3.4.1 Inputs to Task Checking

The sole input to this Function is the Task Table B entry of the Task to be checked.

3.2.1.3.4.2 Task Checking Processing

Task checker interrogates Task B table to determine whether the specific task has been scheduled. If the task has been scheduled then the event set is calculated to determine whether the conditions are met to dispatch the task. If the conditions are met then the Task B table entry for Task status is updated from "Scheduled" to "Dispatched" and any associated activation event is queued on the event queue. Unlatched events are returned to their negated state. The processing of this Function is shown in Figure 3.2.1.3.4.2-1.

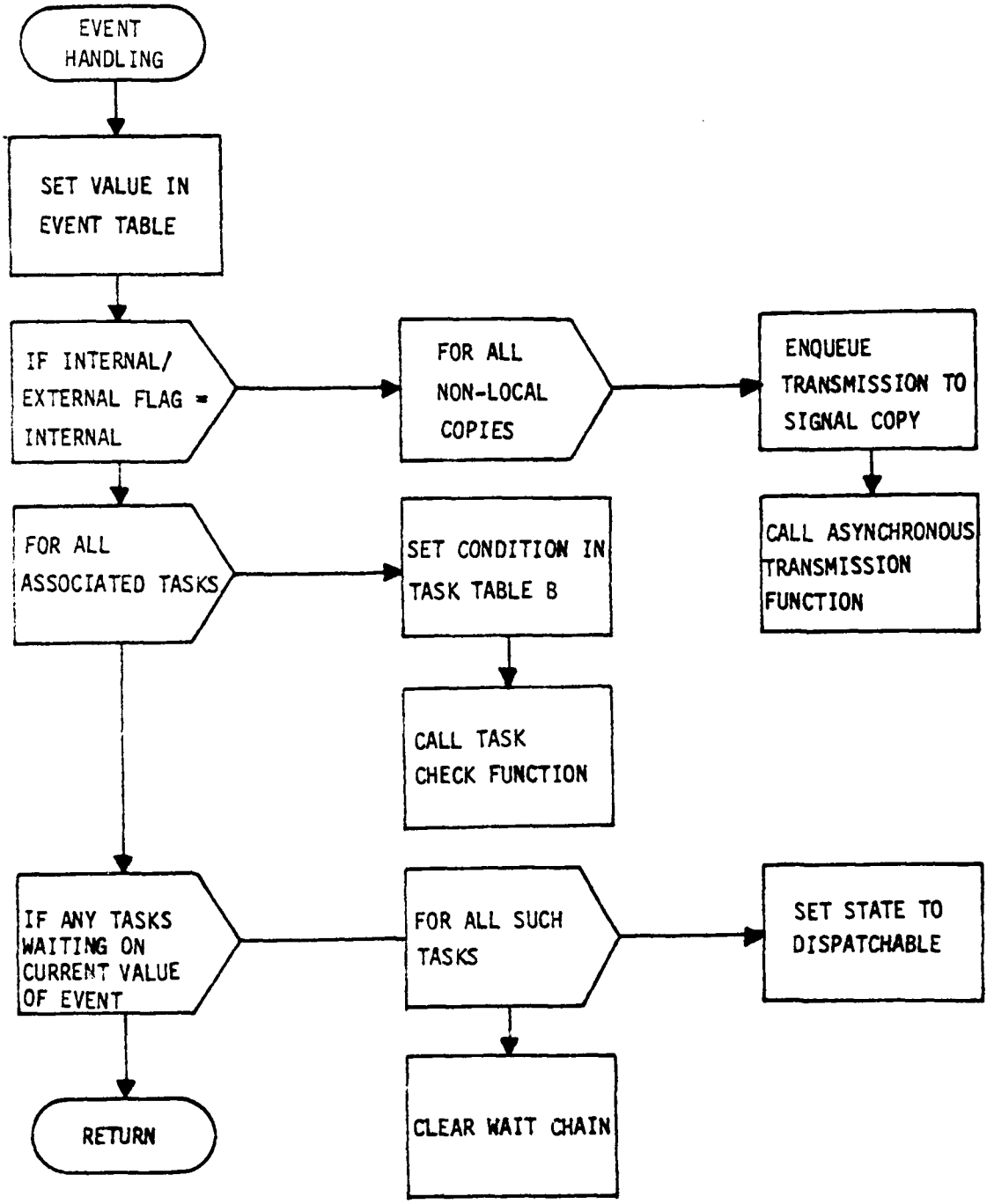


Figure 3.2.1.3.3.2-1. Event Handling Processing

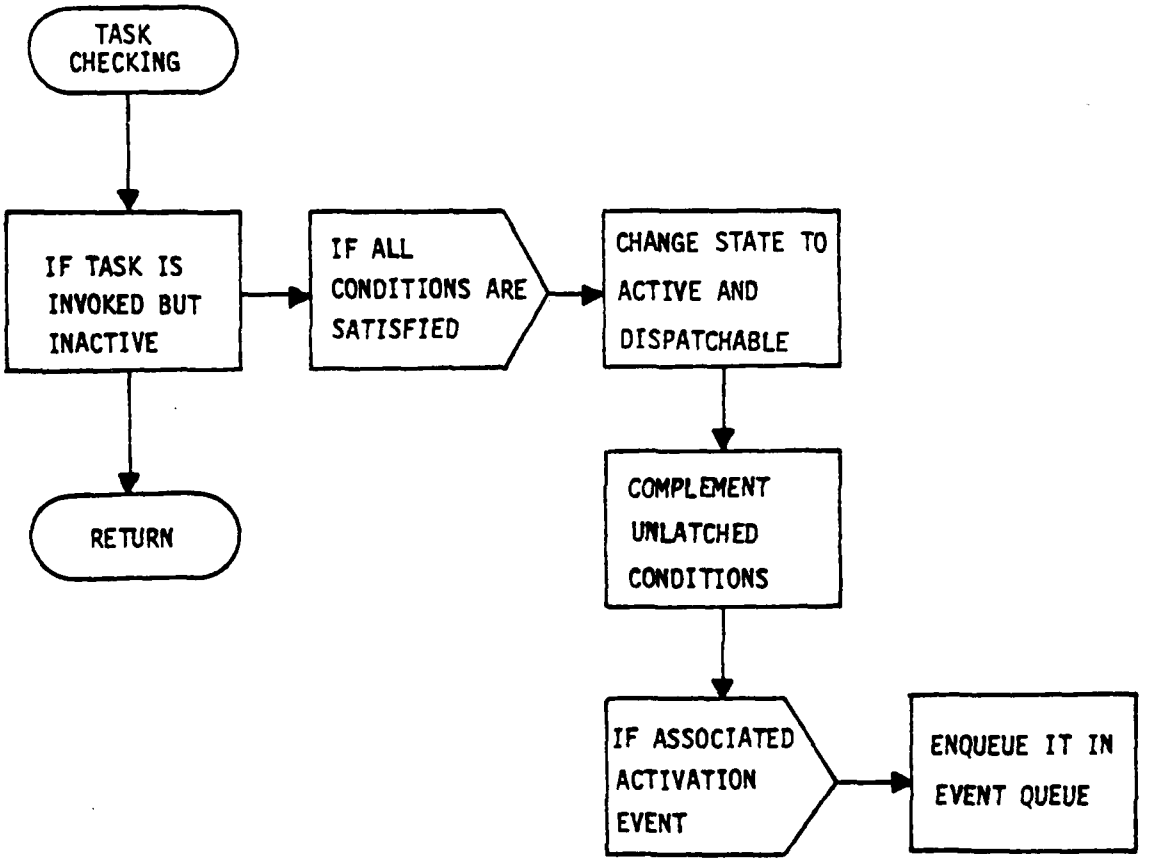


Figure 3.2.1.3.4.2-1 Task Checking Processing

3.2.1.3.4.3 Outputs from Task Checking

The outputs from this Function are:

a. The updated Task Table B entry.
b. The Event Queue, with a request to set the Activation Event, if any, associated with the Task.

3.2.1.3.5 Task Scheduling Function

The Task Scheduling Function is invoked to Schedule a Task, either as the result of the execution of a Schedule Statement within the processor or as the result of a Schedule request received from another processor.

3.2.1.3.5.1 Inputs to Task Scheduling

The sole input to this Function is the Task Table A entry of the Task to be scheduled.

3.2.1.3.5.2 Task Scheduling Processing

The Scheduler uses Task Table A entry to determine if the task is resident in its processor. If the task is resident then Task Table A entry points to the appropriate Task Table B entry which can be set to "scheduled" ("invoked"). The unlatched conditions are reset to the negative of the desired values for dispatching. If the Task Table A entry indicates that the task is in another processor then an asynchronous Schedule Request for that task must be built and submitted for asynchronous transmission. The processing of this Function is shown in Figure 3.2.1.3.5.2-1.

3.2.1.3.5.3 Output from Task Scheduling

The output from this Function is:

If the Task resides in this processor, the updated Task Table B entry; otherwise, the Transmission Queue, with a Schedule request to the processor where the Task resides.

3.2.1.3.6 Task Termination/Cancellation Function

The purpose of the Task Termination/Cancellation Function is to Cancel or Terminate forcibly a specified Task and all of its descendants. If the specified Task is the last dispatched Task, this Function will only affect the Task's descendants, not the Task itself. An additional function is to determine the status of a specific task.

3.2.1.3.6.1 Inputs to Task Termination/Cancellation

The inputs to this Function are:

a. The Task Table A entry of the Task.
b. The identity of the last dispatched Task.
c. The Cancel/Terminate/Invoked Flag.

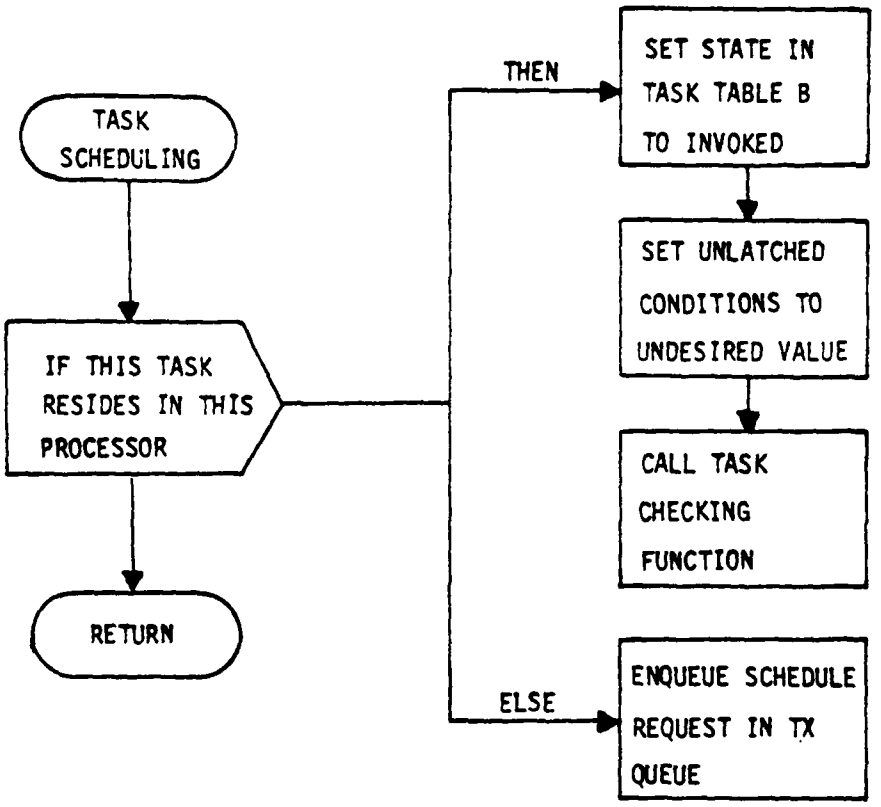


Figure 3.2.1.3.5.2-1 Task Scheduling Processing

3.2.1.3.6.2 Task Termination/Cancellation Processing

The Task Termination/Cancellation performs the following actions:

a. If the specified Task is not in this processor, it enqueues a Cancel or Terminate request in the Transmission Queue and returns.
b. If the specified Task is not the last dispatched Task, it Cancels or Terminates the specified Task.
c. It searches for all descendants of the specified Task whose controllers are in the processor.
d. For each Task found in Step c, if the Task is resident, it Cancels or Terminates it; if the Task is not resident, it enqueues a request to Cancel or Terminate it in the Transmission Queue.

When a Task is Cancelled, its State is set to Uninvoked in Task Tables A and B.

When a Task is Terminated, it is first set Inactive in Task Table B, and then the Task Checking Function is invoked to determine whether to re-Activate the Task. If the Terminated Task has an Activation Event, this Function enqueues a request in the Event Queue to signal the Activation Event off.

In either case, if the Task to be Cancelled or Terminated is in Wait state, it is removed from its Wait chain (see 3.2.1.3.7.2). If the request is to examine the state of a specific task, then the Task Table B Task State entry is returned as a value.

3.2.1.3.6.3 Outputs from Task Termination/Cancellation

The outputs from this Function are:

a. The updated Task Table A and B entries of all Tasks Cancelled or Terminated.
b. The Transmission Queue, with messages to Cancel or Terminate non-local Tasks.
c. Value of the Task State of Execution.

3.2.1.3.7 Wait Function

The Wait Function is used to put a Dispatchable Task into Wait State until a specified time occurs or a specified Event reaches a specified value. See 3.1.2.1.8.4 for further details.

This Function is only invoked by the Application Interface Function.

3.2.1.3.7.1 Inputs to Wait

The inputs to this function are:

a. Time/Event Flag

b. Latched/Unlatched Flag

c. Event table entry or time to go

d. Desired value of event

3.2.1.3.7.2 Wait Processing

The wait conditions are checked to determine whether they have been satisfied and need not wait. Otherwise the Task is placed on either the time wait chain or an event wait chain and the task state in Task Table B is set to wait.

A Wait Chain is a chain of Tasks all waiting for the same type of condition. There is one Wait Chain for Tasks waiting on time, one Wait Chain for each Event waited on, and one Wait Chain for each Event whose complement is waited on.

All Tasks in the same Wait Chain are tied together by the forward and back pointers in their Task Table B entries (see 3.1.2.4.1.14). The Wait Chain on time is ordered by time; all other Wait Chains are ordered arbitrarily. The first Task in a Chain waiting on an Event or the complement of an Event is located by a pointer in the Event Table entry for the Event (see 3.1.2.4.1.3). The identity of the first Task waiting on time is maintained internal to the local Executive.

The processing of this Function is shown in Figure 3.2.1.3.7.2-1.

3.2.1.3.7.3 Outputs from Wait

The outputs from this function are:

a. The Wait Chain into which the Task is put.

b. The updated Task Table B entry of the Task.

3.2.1.3.8 Compool Block Handling Function

This function is responsible for all processing involving Compool Blocks, including:

a. Servicing Read, Write and Trigger Pseudo-Statements.

b. Accepting Asynchronous Compool Block Updates from RT's and other processors.

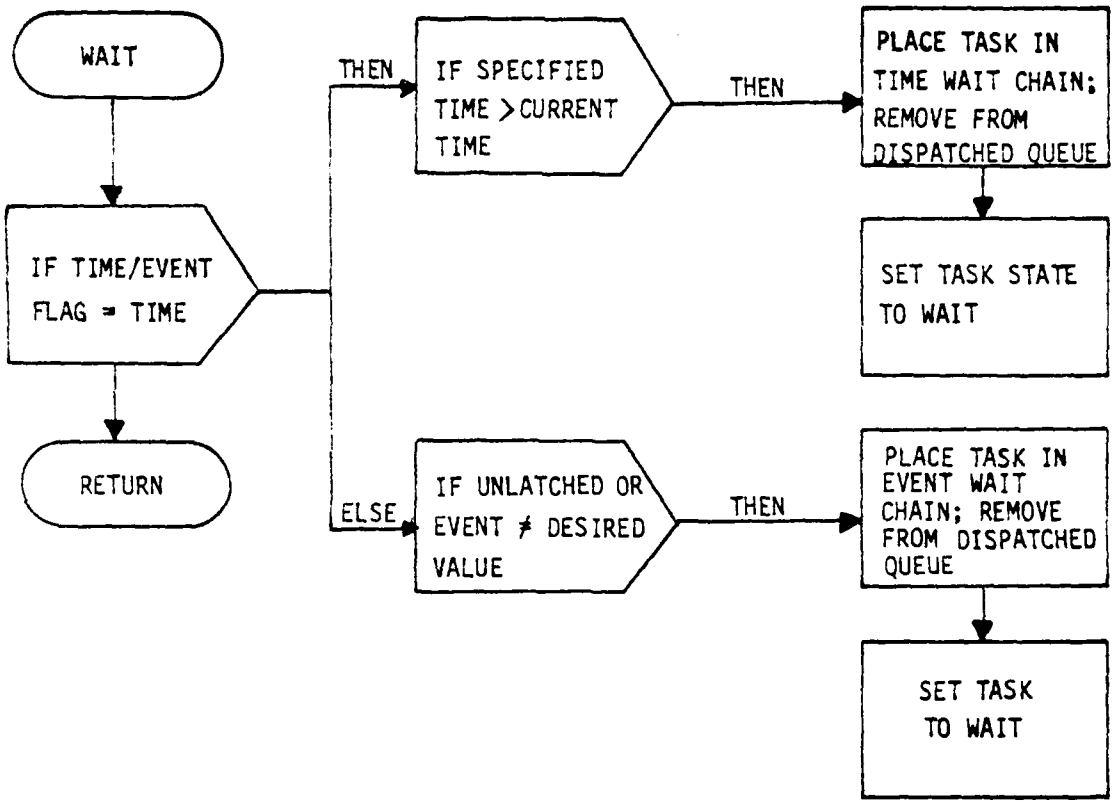


Figure 3.2.1.3.7.2-1 Wait Processing

c. Formulating and enqueuing Asynchronous Messages to update non-local copies of Compool Blocks.

d. Enqueuing Compool Block Update Events to be signalled.

e. In Master Mode, invoking the Master Trigger Function to enqueue Critically Timed Update Messages.

3.2.1.3.8.1 Inputs to Compool Block Handling

The inputs to this Function are:

a. Internal/External Flag

b. Trigger Flag

c. DDB of Compool Block to be processed

d. Compool Block to be processed

e. Local Copy to be processed

f. Time (for Trigger only)

3.2.1.3.8.2 Compool Block Handling Processing

Compool processing involves time dependent compool transmissions as well as normal local-to-global and global-to-local copy transfers, which can involve transfer of compools between processors. The different types of compools are discussed in 3.1.2.1.3.

If the compool is critically timed, then the compool data descriptor block must be updated with the desired transmission time. If the compool is not in the Master Processor then a message must be sent to the master executive requesting transmission at the appropriate time.

If the compool is not critically timed, then local copies must update the global copy upon "write compool" requests and the global compools must update local copies upon "read compool" requests. If compool has been declared as a Global Copy, then there are no local copies in any tasks, i.e., the tasks share the same copy and read and write requests are null functions.

The processing of this function is shown in Figures 3.2.1.3.8.2-1,-2,-3,-4.

3.2.1.3.8.3 Outputs from Compool Block Handling

The outputs from this function are:

a. Compool Block

b. Local Copy of Compool Block

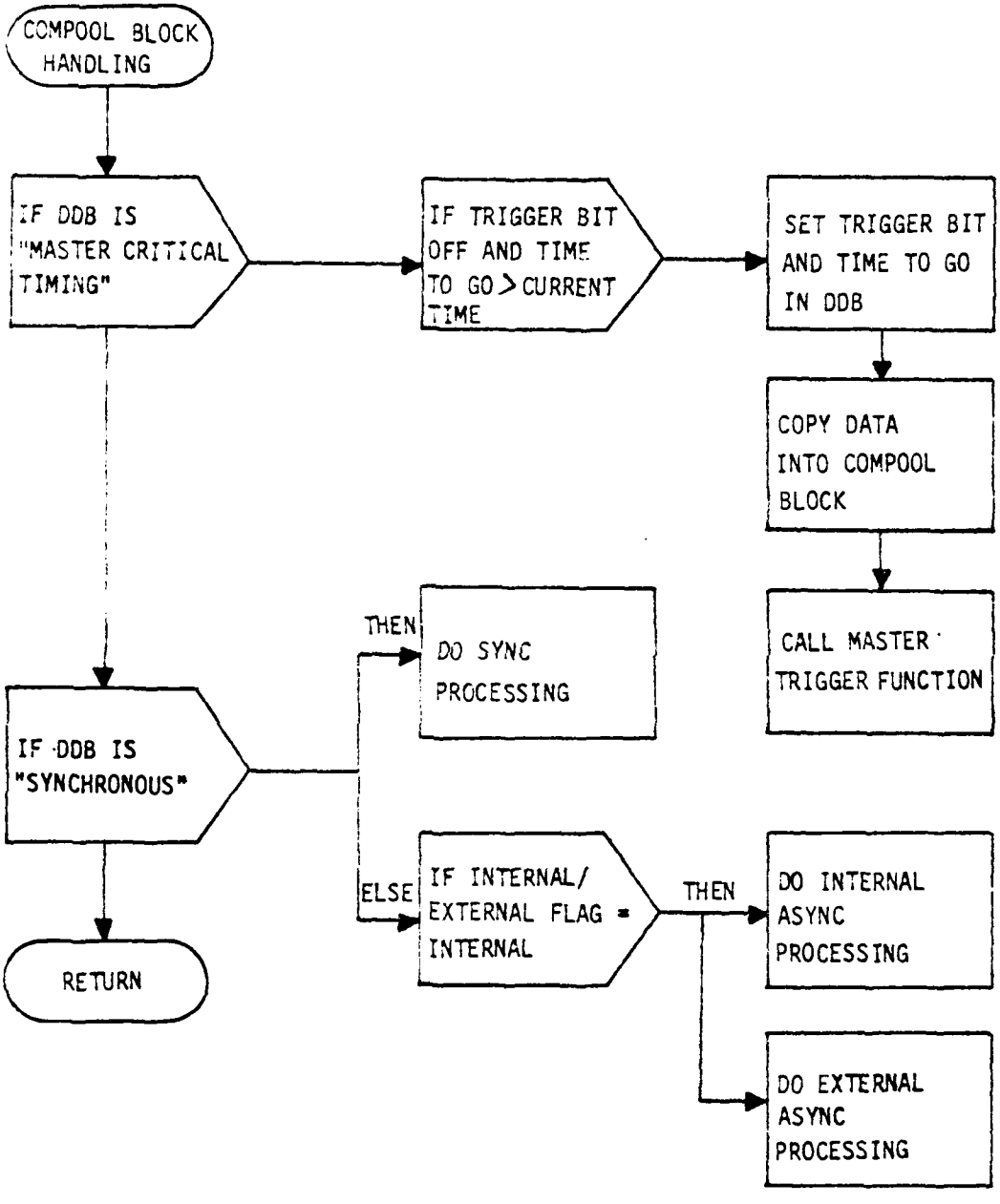


Figure 3.2.1.3.8.2-1 Compool Block Handling

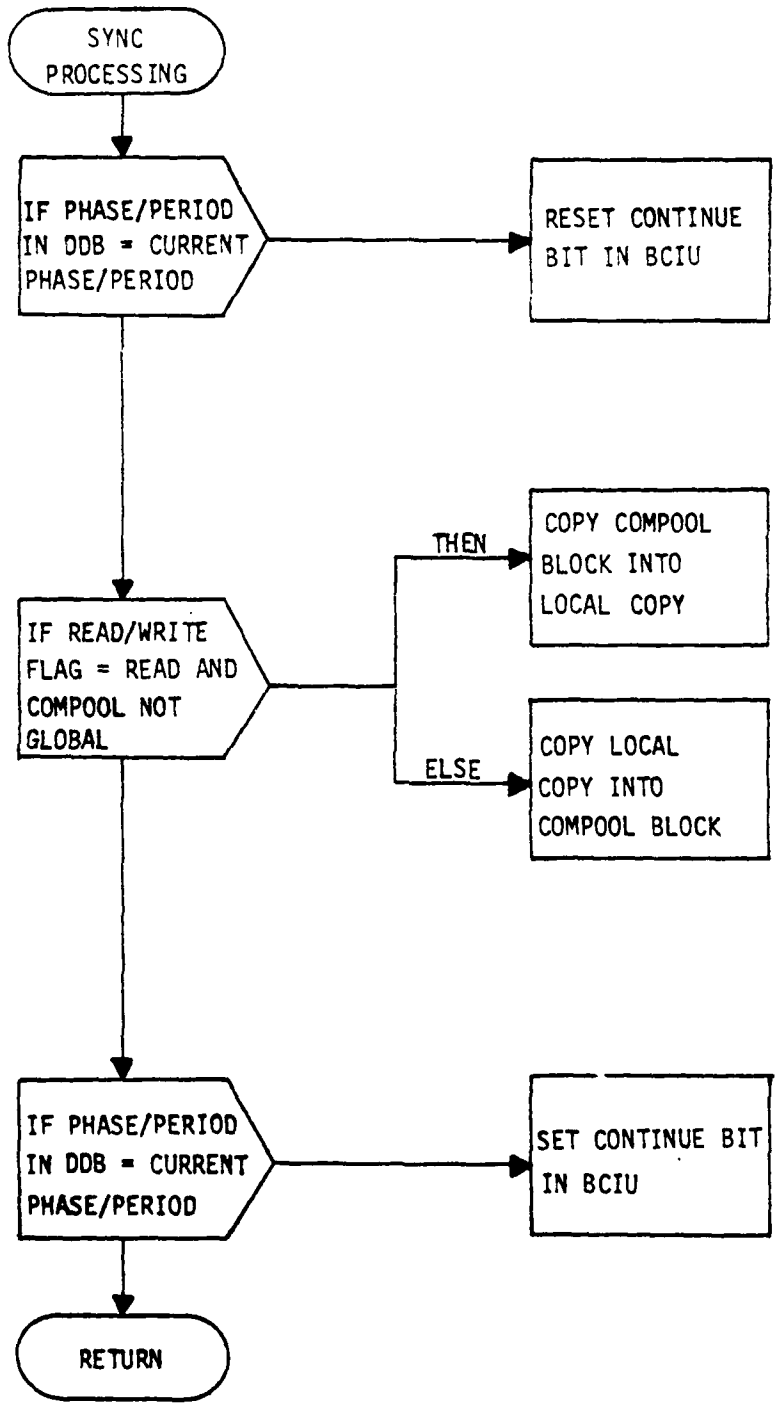


Figure 3.2.1.3.8.2-2 Asynchronous Compool Block Handling

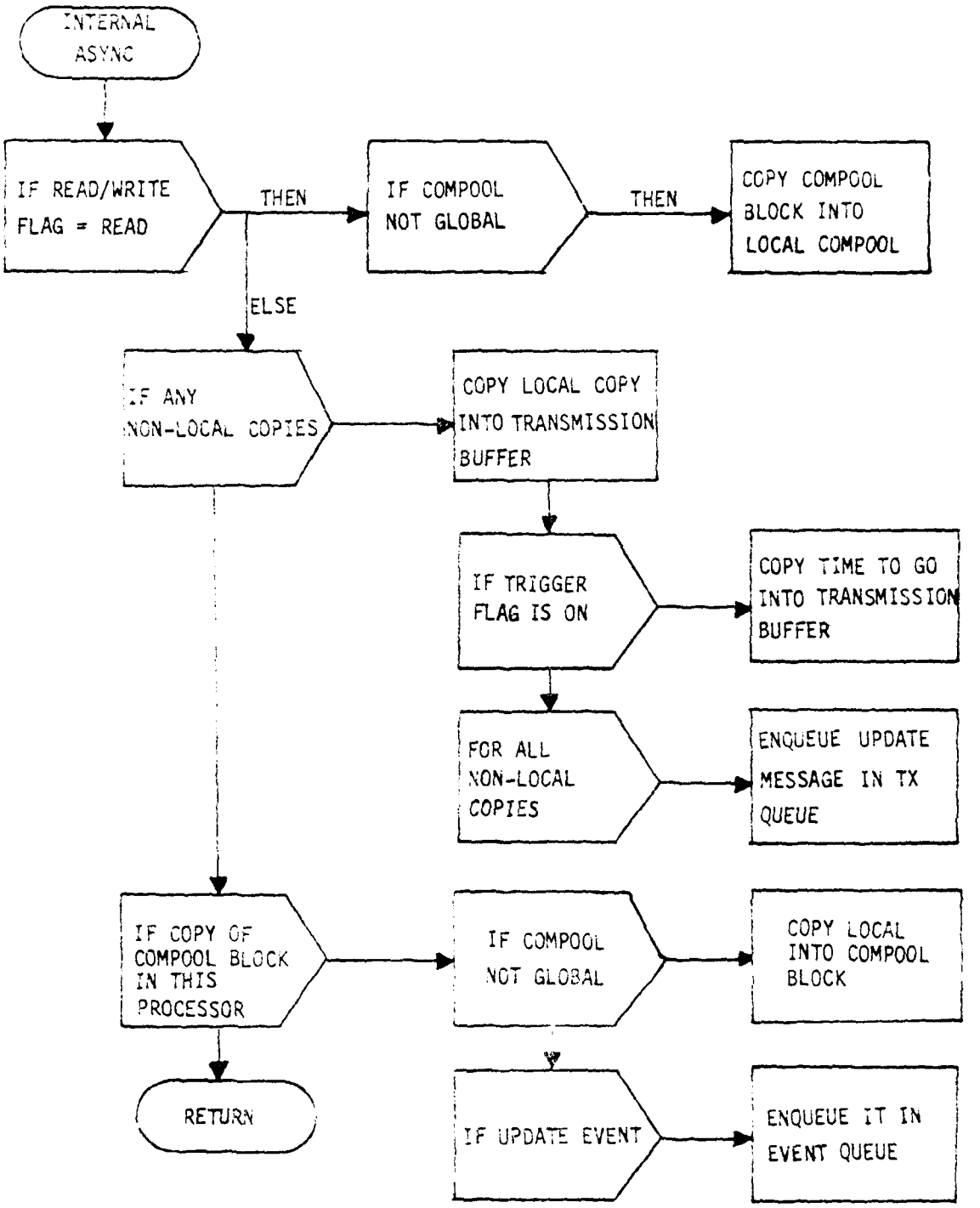


Figure 3.2.1.3.3.2-3 Internal Asynchronous Compool Block Handling

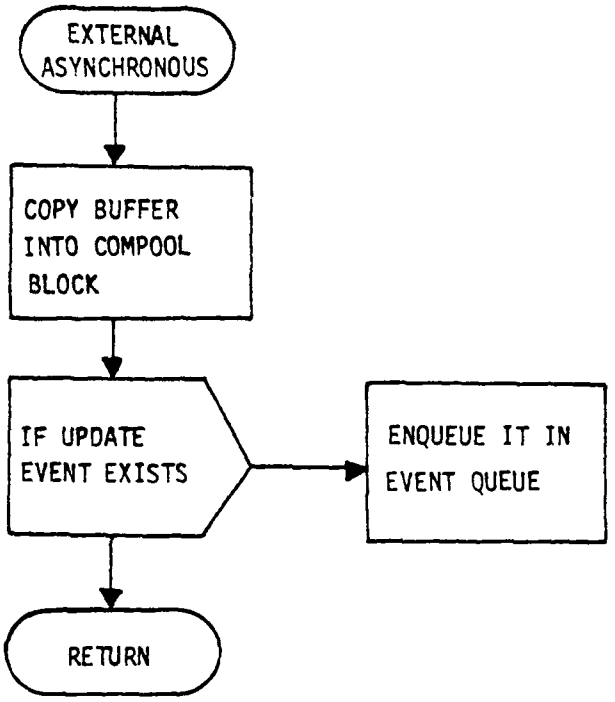


Figure 3.2.1.3.8.2-4 External Asynchronous Compool Block Handling

c. Transmission Queue with enqueued Update messages for copies of Compool Block in other processors.
d. Transmission Queue with enqueued request for Transmission of the critically timed compool at the specified time.
e. Updated DDB (for Master Critical Timing DDB's only)

3.2.1.3.9 Dispatch Function

The Dispatch Function is always and only called by the Local Executive Control Function when the Reception Pending Bit and Minor Cycle Bit are off and the Event Queue is empty. The Dispatch Function searches for the highest priority Dispatchable Application Task and, if it finds one, transfers control of it.

3.2.1.3.9.1 Inputs to Dispatch

The inputs to this Function are:

a. Task Table B
b. The Reception Pending Bit
c. The Minor Cycle Pending Bit
d. The Event Queue

3.2.1.3.9.2 Dispatch Function Processing

When the Local Executive performs any services it keeps track of the Highest Priority task made Dispatchable since the last Dispatch. When the Dispatcher is called it commences scanning of the task table starting with the Highest Priority Dispatchable task. If the last Dispatched task is still the Highest Priority task, then control will be returned to the original task. Otherwise, control will be given to a new task. The only time that the Dispatcher actually searches the task table is when a task ends, and it was the Highest Priority Dispatchable task, or when the Highest Priority Dispatchable task becomes non-dispatchable during the recovery of a condition.

The Dispatch Function performs the following specific actions:

a. When a Privileged Mode Task makes an Executive Service Request, or is interrupted, the Local Executive will always return control directly to that task.
b. It searches Task Table B for the highest priority Dispatchable Task.
c. If none is found, it returns to the Local Executive Control Function.
d. It disables interrupts, and resets the Privileged Mode Flap.
e. If the Task is in Suspended State, it restores the registers, condition status and Comsub Stack Pointer from the Task's Save Area, enables interrupts, and branches to the point where the Task was interrupted.
f. Otherwise, it initializes the Comsub Stack Pointer and calls the Task.
g. On return from the Task, it sets the Privileged Mode Flag.
h. It sets the Task Inactive.
i. If the Task has an Activation Event, it enqueues a request in the Event Queue to signal the Event off.
j. It calls the Task Checking Function to determine whether the Task should be re-activated.
k. It returns to the Local Executive Control Function.

3.2.1.3.9.3 Outputs from Dispatch Function

The outputs from this Function are:

a. Indication of the last Task Dispatched.
b. The Task Table B entry of a Task after it returns to this Function.
c. The Event Queue with a request enqueued to signal the Activation Event of the above Task, if any, off.

3.2.1.3.10 IO Device Function

The IO Device Function provides an interface with the Master Executive. Devices may need to be turned off or be turned on during a particular mission phase. Equipment failures may also require that a malfunctioning unit be shutdown.

3.2.1.3.10.1 Inputs to IO Device Function

The inputs to this function include:

a. Device Number
b. On-Off Flag
c. Reconfiguration Flag

3.2.1.3.10.2 IO Device Processing

The processing consists of the formulation of a message to the Master Executive with the same contents as the inputs. If the application task is in the master processor then a call to the Command List Handler.

3.2.1.3.10.3 Outputs from IO Device

The output is the message formulated in the processing.

3.2.1.4 Initialization and Recovery Function

The Initialization and Recovery Function is divided into the following subfunctions:

a. The Initialization and Re-Initialization Function
b. The Local Executive Error Recovery Function
c. The Power Down Function

3.2.1.4.1 Initialization and Re-Initialization Function

The Initialization and Re-Initialization Function is automatically invoked by the hardware upon system initialization or recovery from a power failure. It is responsible for initializing of re-initializing the state of the Local Executive.

3.2.1.4.1.1 Inputs to Initialization and Re-Initialization

The inputs to this Function are:

a. The Power Down Flag
b. The prior state of the system

The Power Down Flag, if on, indicates that a successful Power Down has been accomplished.

3.2.1.4.1.2 Initialization and Re-Initialization Processing

The Initialization and Re-Initialization Function performs the following steps:

a. If the Power Down Flag is on, it restores the state prior to the power failure.

b. Otherwise, it examines the state of the system and determines what to initialize or re-initialize.

The BCIU initialization sequence and interaction with the processor are described below. At the time that power is applied (cold start or transient recovery), the Bus Control Module shall clear the Processor Control Resister (PCR) and its Internal Status Register (ISR) and perform any other initialization required. The BCM shall then perform its power-on self-test and set the READY BIT within the PCR to logic 1 and present the Power-On Initialization Interrupt (Level 1 with ISR zero) to the Processor. The FAIL Bit within the PCR shall be set according to the self-test results. A logic 1 shall indicate that the self-test failed. In either case, the BCM shall begin to monitor the GO Bit within the PCR. When the BCM detects that the GO Bit has been set to logic 1 by the Processor, the BCM shall assume that the Processor has loaded the BCIU Address into the PCR Bits 7-11. This address shall be saved in a non-PIO accessible register and shall be the BCIU's Address.

The BCM shall then enter the Quiescent Mode. This shall be the normal entry point from either Master or Remote Mode. When the BCM detects that the GO Bit of the PCR has been set to 1 by the Processor, the BCM shall set the Run Bit of the PCR to 1 and examine the Master bit of the PCR. If the Master bit of the PCR is 1, the BCM shall proceed to operate in the Master Mode. If the Master bit is 0, the BCM shall proceed to operate in the Remote Mode. If, during the course of operating in either mode, the BCM discovers the GO bit of PCR has been set to 0 by the Processor, the BCM shall complete the bus operation that it is presently performing (if any) and return to the Quiescent Mode entry point.

After initialization or re-initialization, the following should be true:

a. The BCIU should have the proper Terminal Address, and should be running.

b. The Reception, Event, Minor Cycle and Transmission Queues should all be empty.

c. DMA Pointer Block 0 should be set up for Minor Cycle 0.

d. The Minor Cycle Number = 0.

e. The various areas of core should be Write Protected properly.

f. All global parameters, such as the number of processors, should be initialized.

After the system is fully initialized, this Function transfers control to the Local Executive Control Function.

3.2.1.4.1.3 Outputs from Initialization and Re-Initialization

The outputs from this Function are:

a. Global Executive parameters

b. The state of the BCIU

3.2.1.4.2 Local Executive Error Recovery Function

This Function is invoked upon detection by the hardware or the Local Executive of an unrecoverable error within the processor or BCIU.

3.2.1.4.2.1 Inputs to Local Executive Error Recovery

The sole input to this Function is indication of the condition causing the failure. These conditions shall include at least:

a. Illegal operation code

b. Boundary alignment error

c. Processor parity error

d. Processor memory protect

e. DMA parity error

3.2.1.4.2.2 Local Executive Error Recovery Processing

The Local Executive goes into the halt state after setting the status register in the BCIU.

3.2.1.4.2.2 Outputs from Local Executive Error Recovery

The sole output from this function is a status code to the BCIU indicating processor failure.

3.2.1.4.3 Power Down Function

This Function is invoked upon detection of a power failure. It attempts to save the state of the processor prior to total failure.

3.2.1.4.3.1 Inputs to Power Down

The input to this Function is the state of the processor at the time of the power down interrupt.

3.2.1.4.3.2 Power Down Processing

This Function attempts to save the registers at the time of the power down interrupt. If it is successful, it then sets the Power Down Flag.

3.2.1.4.3.3 Outputs from Power Down

The outputs from this Function are:

a. The state of the processor at the time of failure

b. The Power Down Flag

3.2.2 Master Executive Functions

The Master Executive includes the following major functions:

a. Master Initialization Function

b. Master Time Control Function

c. Master Synchronous Control Function

d. Master Asynchronous Control Function

e. Master Error Recovery Function

f. Mass Memory Control Function

3.2.2.1 Master Initialization

The Master Initialization Function provides for initialization of the IDAMST System. It loads the Remote and Monitor Processor from Mass Memory and performs initial testing of the system.

3.2.2.1.1 Inputs to Master Initialization

The Inputs to Master Initialization are:

a. The Initial Program Load performed by the hardware bootstrap procedure. This contains all the executive tables created as part of compiling and loading the system.

b. The number of the processor containing the Master Executive. This is supplied by a hardware discrete which can be attached to the processor.

c. The Mass Memory containing the object modules for this mission.

3.2.2.1.2 Master Initialization Processing

3.2.2.1.2.1 Initial Step

The Master Processor is determined by a discrete which indicates no time delay before attempting to load. The remaining processors will have different lengths of time to wait before attempting to load as the master processor.

After the Master Processor is loaded and has started execution, the Master Processor:

a. Determines its Processor Number. This number indicates the number of processors that failed to load and can be eliminated from the load sequence.

b. Sets its BCIU to Master Mode and sets its BCIU number.

c. Attempts to communicate with the next processor or the Master Processor may be unable to communicate with any other processor, in which case the Master Processor will load the Monitor Processor System into the same processor in which the Master resides.

The Master may be able to communicate with at least one other processor, in which case this processor is designated as Monitor. The Monitor System is loaded into this processor.

If the Master is able to communicate with the other two processors, each processor is loaded with its appropriate software (Local Executive and applications routines).

As part of the communication, the Master Executive indicates to the other processors that they should not try to load on the Master Executive.

3.2.2.1.2.2 Normal Start-Up

Normal Start-Up commences if all three processors are participating in the system.

The Master sends the Monitor a System Interrupt Command. This causes the Monitor to be initialized. The Master then sends the first Minor Cycle Event (Master Function Mode Command) to each processor. This awakens each Local Executive and causes the Local Executive to perform initialization and prepare for receiving/transmitting data for the first minor cycle. When the Local Executive has completed this function, it invokes the Master Executive to schedule the Master Sequencer which starts all the Application Tasks.

3.2.2.1.2.3 Abnormal Start-Up With Less Than 3 Processors

The Master Processor is loaded for the three processor configuration. After loading, the Master is able to determine the available complement of processing elements. If the full complement is not available, then the Master must search the Mass Memory to find the appropriate configuration to load. The Master processor is then reloaded with the proper Master Processor memory. The Normal Startup procedure for loading and starting the remaining processor (if there is one) is followed.

3.2.2.1.3 Outputs of Master Initialization

The outputs of Master Initialization are:

a. The Master Processor containing the Master Executive, a Local Executive, and some applications software.

b. The Monitor Processor loaded with the Monitor, a Local Executive, some Applications Software, and the applications modules needed for a limited mission.

c. The Remote Processor loaded with Local Executive and the applications software for the mission.

d. The scheduling of the Master Sequencer.

3.2.2.2 Master Time Control Function

The Master Time Control Function consists of three subfunctions:

a. Timer B Control Function

b. Timer A Control Function

c. Master Trigger Function

3.2.2.2.1 Timer B Control Function

This function is involked upon an interrupt by Timer B (see 3.1.1.2.4). This indicates that Timer B has reached the value of zero. Since Timer B is a 16-bit count, incremented every 100 microseconds, and is never set after system initialization, this Function is invoked every 6.5536 seconds.

The sole purpose of this Function is to keep track of the passage of absolute time. At any point, absolute time is defined as one hundred microseconds times the value of Timer B plus the time since the last timer B interrupt.

3.2.2.2.1.1 Inputs to Timer B Control

The inputs to this Function are:

a. The Timer B interrupt

b. The time of the last Timer B interrupt

c. Current Timer B value

3.2.2.2.1.2 Timer B Control Processing

The Timer B Control Function adds 6.5536 seconds to "time of last Timer B interrupt." If invoked by a request for elapsed time, the value of the timer is added to the cumulative value.

3.2.2.2.1.3 Outputs from Timer B Control

The sole output from this Function is the updated time of the last Timer B interrupt.

3.2.2.2.2 Timer A Control Function

The Timer A Control Function is invoked upon an interrupt by Timer A. This indicates the expiration of an interval of time determined by the previous invocation of the Timer A Control Function. This interval will have been set to expire upon either or both of the following conditions:

a. Time for a new Minor Cycle

b. Time to send a Critically Timed message

3.2.2.2.2.1 Inputs to Timer A Control

The inputs to this Function are:

a. The Timer A interrupt

b. Absolute time (see 3.2.2.2.1)

c. The Critically Timed Message Queue (see 3.2.2.2.3.1)

d. The status of Synchronous operations (see 3.2.2.3.3)

e. The previous theoretical Minor Cycle number

Note that this Function maintains only the theoretical Minor Cycle number. The actual Minor Cycle number is set by the Master Synchronous Control Function, and may lag behind the theoretical Minor Cycle number due to an exceptionally heavy Bus loading.

3.2.2.2.2.2 Timer A Control Processing

The Timer A Control Function performs the following actions:

a. It checks the Critically Timed Message Queue to see if there are any Critically Timed Messages ready to transmit.

b. If so, it sends them.

c. It checks to see whether any Critically Timed Messages should be sent before the next Minor Cycle. If so, it sets Timer A to expire at the time to send the next Critically Timed Message; if not, it sets Timer A to expire at the time for the next Minor Cycle.

d. It invokes Synchronous Control of time for a minor cycle has expired.

3.2.2.2.2.3 Outputs from Timer A Control

The outputs from this Function are:

a. The updated Critically Timed Message Queue

b. Any Critically Timed Messages set to go at this time

c. The new value for Timer A

3.2.2.2.3 Master Trigger Function

The Master Trigger Function processes Critically Timed Messages detected by the Compool Block Handling Function (see 3.2.1.3.8). It enqueues them in the Critically Timed Message Queue for processing by the Timer A Control Function.

3.2.2.2.3.1 Inputs to Master Trigger

The inputs to this Function are:

a. The Critically Timed Message

b. Time to send the Critically Timed Message

c. The Critically Timed Message Queue

The Critically Timed Message Queue contains all Critically Timed Messages which have been Triggered but not yet sent to the appropriate RT. The Queue is arranged in order of the time at which the messages are to be sent. The items in the Queue are Master Critical Timing DDBs (see 3.1.2.4.1.7). They are linked together by item #6, "Forward Pointer to DDB." The identity of the first DDB in the Queue is maintained local to the Master Executive.

3.2.2.2.3.2 Master Trigger Processing

The Master Trigger Function inserts the new Critically Timed message into the proper place in the Critically Timed Message Queue.

3.2.2.2.3.3 Outputs from Master Trigger

The sole output from this Function is the updated Critically Timed Message Queue.

3.2.2.3 Master Synchronous Control Function

The Master Synchronous Control Function controls the Synchronous operations of the Master BCIU. It may be invoked either by the Timer A Control Function at the time for a new Minor Cycle, or by a Program controlled Interrupt generated by the BCIU when it has finished processing the Synchronous Command List.

3.2.2.3.1 Inputs to Master Synchronous Control

The inputs to this Function are:

a. The actual Minor Cycle number

b. The theoretical Minor Cycle number

c. The prior state of Synchronous operations

d. Current state of Synchronous operations

3.2.2.3.2 Master Synchronous Control Processing

The Master Synchronous Control Function performs the following actions:

a. If invoked by the Timer A Control Function and all synchronous operations are complete, then increment the theoretical Minor Cycle and go to step 'e.'

b. If invoked by the Timer A Control Function and all synchronous operations are not complete, then increment the theoretical Minor Cycle Number and return.

c. If invoked because of the BCIU interrupt which indicates the end of synchronous processing and the theoretical Minor Cycle = actual Minor Cycle, then begin processing asynchronous transmission list, if non-empty.

d. If invoked because of the BCIU interrupt which indicates the end of synchronous processing and the theoretical Minor Cycle is greater than the actual Minor Cycle, then go to step 'e.'

e. It increments the actual Minor Cycle number by one, and sets the Minor Cycle Pending Bit in the Master processor (see 3.2.1.1.3).

f. It links together the proper blocks of Instructions in the Synchronous Command List for the new Minor Cycle via the Command List Handler (see 3.2.2.4).

g. It sends Master Function Mode Commands to the Remote processors to inform them of the new Minor Cycle.

h. It sets the Master BCIU to the beginning of the Synchronous Instruction List via the BCIU Interface Function (see 3.2.2.5).

3.2.2.3.3 Outputs from Master Synchronous Control

The outputs from this Function are:

a. The new Minor Cycle number

b. The theoretical Minor Cycle

c. The current state of Synchronous operations

3.2.2.4 Command List Handler Function

The Command List Handler is responsible for the control and modification of the command list that controls the BCIU. Those functions include turning on/off communication to and from devices, setting the commands to match the minor cycle number, and insert synchronous messages into the command list.

3.2.2.4.1 Command List Handler Inputs

Inputs to the Command List Handler include:

a. Device number

b. whether communication with the device should be turned on or off

c. whether the command list should be modified because of reconfiguration

d. minor cycle number

e. asynchronous message DDB

f. append/insert/In-Out/Link List Flag

3.2.2.4.2 Processing

Four processing subfunctions are identified as part of the major functions: In/Out, Link List, Insert Message, and Append Message of communication with a device is to be altered, then the Command List Handler in the form of sub-function In/Out must determine the location of the device in all of the command lists.

A device operation needing alteration may only require one change in the command list or, in the case of the bus, every command may require alteration to indicate a change in bus operation. Reconfiguration to a one processor system requires that the master processor command list be shortened to the minimal communication list.

If the command list must be changed because of the beginning of a new minor cycle, then the BCIU pointer to the appropriate command list must be altered. Command Handler in the form of Link List must match the minor cycle number to the appropriate command list, and then present the request to change the BCIU pointer to the BCIU interface.

If an asynchronous message must be sent the message can either be inserted into the message list for immediate transmission or the message can be postponed until the end of the synchronous transmissions. The Command Handler in the form of Insert message and Append Message is responsible for the manipulation of these lists.

3.2.2.4.3 Command List Outputs

The output will be an updated command list.

3.2.2.5 BCIU Interface Function

The BCIU Interface module is responsible for setting and reading the BCIU registers via the programmed I/O operations. These functions serve as the interface between the hardware unit and the Master Executive Software.

3.2.2.5.1 BCIU Interface Inputs

Inputs to this module will consist of requests to read a particular BCIU register or to write a register, along with the value to be written into the register (see 3.1.1.1.3.2). An additional input is the interrupt level for a BCIU generated interrupt.

3.2.2.5.2 BCIU Interface Processing

The BCIU Interface will generate the PIO operation to any specific register accessible by the processor. If an interrupt level is presented to the BCIU Interface, then the Interface module will interrogate the specific BCIU registers to determine the precise meaning of the interrupt and call the

appropriate service module (e.g., Error Recovery).

3.2.2.5.3 BCIU Interface Outputs

Outputs of BCIU Interface will be the values of the BCIU registers that are requested to be changed or to be read.

3.2.2.6 Master Asynchronous Control Function

The Master Asynchronous Control Function responds to Asynchronous Request Vectors received either from other processors over the Data Bus or from the Local Executive within the Master Processor.

3.2.2.6.1 Inputs to Master Asynchronous Control

The inputs to this Function are:

a. The Request Vector

b. The Master Request Decode Table (see 3.1.2.4.2.1)

c. The status of Synchronous operations

3.2.2.6.2 Master Asynchronous Control Processing

The Master Asynchronous Control Function determines which command to send to the Master BCIU by using the Request Vector to index into the Master Request Decode Table. The Master Request Decode Table will indicate whether the message will be sent out immediately or be queued for transmission following the completion of the Synchronous Command List. If the BCIU is processing Synchronous Commands and the request is for immediate transmission, this Function links the Asynchronous Instruction into the Synchronous Instruction List; if the BCIU has completed the Synchronous List, it simply sends the Instruction to the BCIU.

If the request for transmission comes from an RT, this Function will perform the same actions as described above, except that it will use the Master Remote Terminal Request Tables (see 3.1.2.4.2.3) rather than the Master Request Decode Table to identify the request.

3.2.2.6.3 Outputs from Master Asynchronous Control

The outputs from This Function are commands to the Master BCIU via the Command List Handler.

This page left blank intentionally.

3.3 ADAPTATION

This section summarizes the IDAMST Executive requirements with respect to the operating facility, system parameters, and the internal capacities of the system itself.

3.3.1 General Environment

The IDAMST Executive must be able to run in two environments:

a. DEC-10 Mode (SLS)
b. IDAMST Processor Mode (ICS, STS and ITB)

In IDAMST Processor Mode, the Executive will execute native code, and communicate with real or simulated Remote Terminals over a real or simulated Data Bus.

In DEC-10 Mode, on the other hand, the Executive will execute PDP-10 code, and the operations of the Data Bus, RTs, Timers and interrupts will have to be replaced by logically parallel simulated constructs compatible with the conventions of the DEC-10.

3.3.2 System Parameters

There are two constant referenced by the IDAMST Executive which may change according to operation needs:

a. The number of processors

b. The rates of the Minor Cycle and Major Frame

The number of processors in the IDAMST federated system may vary from one to sixteen. The IDAMST System must include enough processors to supply the computing power necessary to support the desired application. On the other hand, the overhead associated with intertask communication tends to increase as Tasks are partitioned into more processors.

The Minor Cycle rate and number of Minor Cycles per Major Frame may be changed to suit the requirements of the peripheral equipment and the computational algorithms. Currently, it is anticipated that there will be one Major Frame per second and 64 Minor Cycles per Major Frame.

3.3.3 System Capacities

Since all Application Software entities are controlled from table entries preallocated by PALEFAC, the capacity of the IDAMST Executive is essentially limited only by available memory. The sole exception to this is the three queues used by the Local Executive: The Transmission Queue, the Reception Queue, and the Event Queue and the one queue of the Master Executive: the Postponed Asynchronous Transmission Queue. When any of these queues becomes full the system degrades. Therefore, it is necessary that these queues be allocated long enough to handle the maximum expected loading.

4.0 QUALITY ASSURANCE PROVISIONS

This section identifies the basic method for accomplishing software verification.

4.1 Introduction

IDAMST CPCIs will incorporate top-down, structured concepts, described briefly below:

Structured Program

A structured program is a computer program constructed of a basic set of control logic figures which provide at least the following: Sequence of two or more operations, conditional branch to one of two operations and return repetition of an operation. A structured program has only one entry and one exit point. A path will exist from the entry to each node and from each node to the exit. In addition, certain practices are associated, such as indentation of source code to represent logic levels, use of intelligent data names and descriptive commentary.

Top-Down Programming

Top-down programming is the concept of performing in hierarchical sequence a detailed design, code, integration and test as concurrent operations.

Top-Down Structured Programs

A top-down structured program is a structured program with the additional characteristics of the source code being logically but not physically segmented in a hierarchical manner and only dependent on code already written. Control of execution between segments is restricted to transfers between vertically adjacent hierarchical segments.

Top-down coding and verification is an ordering of system development which allows for continual integration of the system parts as they are developed and provides for interfaces prior to the parts being developed. At each stage, the code already tested drives the new code, and only external data is required.

In top-down programming, the system is organized into a tree structure of segments. The top segments contain the highest level of control logic and decisions within the program, and either passes control to the next level segments or identifies the next level segments for in-line inclusions. The next level may include stubs. Stubs which are to be replaced eventually with running code may contain a "no operation" instruction or possibly a display statement to the effect that control has been received. The process at replacement of successively lower level stubs with operational code continues until all functions within a system are coded and verified.

In top-down coding and verification, the highest level element is coded first. Coding, checkout, and integration proceed down the hierarchy until the lowest levels have been integrated. This does not imply that all elements at a given level are developed in parallel. Some branches will intentionally be

developed early, e.g., to permit early training and early development of critical functions or hardware/software integration.

Many systems interfaces occur through the data base defintion in addition to calling sequence parameters. Top-down programming requires that sufficient data definition statements be coded and that data records be generated before exercising any segment which references them. Ideally, this leads to a single set of definitions serving all the programs in a given application.

This approach provides the ability to evolve the product in a manner that maintains the characteristic of always being operable, extremely modular and always available for successive levels of testing that accompany the corresponding levels of implementation. Exception to the top-down coding and integration approach will be considered on a case-by-case basis.

Each computer program will be coded in a higher order language. Use of assembly or machine language will be restricted to coding of certain executive functions where the higher order language cannot be used.

Real Time Structured Programs

An additional complexity in the IDAMST system is the Real Time, asynchronous communication of structured programs as tasks. Tasks are also organized as a hierarchy. Each task has a Controller Task which is the only task permitted to schedule or cancel the lower level task. However, any task is permitted to activate any other task in IDAMST.

4.2 Computer Program Verification

Computer program verification is the process of determining whether the results of executing a computer program in a test environment agree with the specification requirements. Verification is usually only concerned with the logical correctness of the computer program (i.e., satisfying the functional/performance requirements) and may be a manual or a computer-based process (i.e., testing software by executing it on a computer).

The use of top-down structured programming techniques provide certain program characteristics that may lead to a simplification of the computer program verification process. Top-down integration of the program elements in a CPCI minimizes the use of complex driver routines and replaces them with actual program elements and simple program stubs. It also provides a system in which the computer program is continually being tested as successively lower levels of program elements are integrated and the interfaces between program elements are verified prior to the integration of the next lower level.

4.2.1 Program Element Tests

Program elements are coded in the sequence required for top-down integration. When coding and code review are completed, each program element shall be functionally tested in a stand-alone configuration by the programmer to assure that the element can be executed and that the specified functions are performed. Since program elements are small and are restricted to one entry point and one exit point, the test environment is relatively simple.

4.2.2 CPCI Integration Tests

Following successful completion of the Program Element Tests, the program elements are entered into the Computer Program Library where they are subjected to configuration control procedures. Controlled program elements are compiled/assembled, link-edited and the current CPCI version is made available for integration testing. Integration tests are dynamic tests designed to verify program functions and interfaces between program elements and with the data base. The result is a complete CPCI for which all design features have been verified.

The integration of program elements or tasks into the complete computer program shall be accomplished in a top-down sequence. The highest level elements which contain the highest level controller tasks shall be tested and integrated first. These tasks are the Master Sequencer, Configurator, Request Processor, and Subsystem Status Monitor. Testing and integration shall proceed down the hierarchy until all program elements (e.g., equipment interface functions), have been integrated and the design completely verified.

An important aspect of integration testing of IDAMST will be the invocation and synchronization of the tasks, since these functions do not fall under the structured programming rules.

4.2.3 Formal Software Testing

The purpose of formal testing is to confirm that the computer program performs the functions and satisfies the performance requirement contained in the software requirements specification. Formal testing consists of Preliminary Qualification Tests (PQT) and Formal Qualification Tests (FWT), and are conducted in accordance with Air Force approved test plans.

Pre-Qualification Testing (PQT)

PQT is an incremental process which provides visibility and control of the CPC2 development during the time period between the Critical Design Review and Formal Qualification Testing.

PQT consists of functional level tests, conducted at the development facility, and using Air Force approved test plans. These tests will use documented procedures, completed by the contractor, and submitted to the Air Force Sufficiently in advance of the scheduled test session to permit review and analysis. They will typically use controlled inputs specifically prepared for the test purpose.

A Pre-Qualification test will generally be conducted for each CPCI function. If a test's cost or time consumption estimates are significantly high, the test will be deferred to FQT unless it is time-critical or performance-critical to the development of the CICI.